CLASSES OF ARRANGEMENT GRAPHS IN THREE DIMENSIONS

Elspeth J. Nickle
B.Sc, University of Lethbridge, 2002

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

Abstract

A 3D arrangement graph G is the abstract graph induced by an arrangement of planes in general position where the intersection of any two planes forms a line of intersection and an intersection of three planes creates a point. The properties of three classes of arrangement graphs — four, five and six planes — are investigated. For graphs induced from six planes, specialized methods were developed to ensure all possible graphs were discovered. The main results are: the number of 3D arrangement graphs induced by four, five and six planes are one, one and 43 respectively; the three classes are Hamiltonian; and the 3D arrangement graphs created from four and five planes are planar but none of the graphs created from six planes are planar.

Acknowledgements

First, I wish to thank my supervisor, Stephen Wismath, for his encouragement, guidance, and kindness throughout the process. I especially appreciated his unstinting patience, the valuable discussions and his many helpful insights during the writing phase. I am also grateful to my co-supervisor, Daya Gaur, for his useful suggestions and his exceptional care and attention in reviewing my work

In addition, I wish to extend a special thanks to the University of Lethbridge in general, and the Department of Mathematics and Computer Science in particular, for a very positive and supportive environment in which to study. I also enjoyed my fellow students in the Computational Geometry Laboratory who never ceased to provide fun, interest and help when needed.

Finally, but far from least, I am deeply indebted to my husband, Ron Teather, for his inexhaustible support, his unwavering love — and for putting up with all my erratic hours.

Lethbridge, July 2005
Elspeth Nickle

Contents

List of Tables

List of Figures

Chapter 1

Introduction, Previous Research and Research Goal

1.1 Introduction

The goal of this thesis is to study the properties of 3-dimensional (3D) arrangement graphs, a class of graphs induced by arbitrary arrangements of planes. Given a set of planes in general position, where no four planes intersect at a common point and no pair of planes are parallel, the intersection of any two planes then forms a line of intersection, and an intersection of three planes creates a point. Informally, as shown in the representation in Figure 1.1, an arrangement graph is created with the vertices represented by the points, and the edges represented by the segments of the lines of intersection between two intersection points. The simplest instance of an arrangement graph is a tetrahedron formed by four planes. An instance of the next simplest, an arrangement graph formed by five planes, is shown in Figure 1.1.

While a body of work exists on the properties of 2-dimensional (2D) arrangement graphs (those induced by a set of lines in the plane instead of a set of planes in 3-space), to the best of our knowledge no corresponding body of work exists regarding 3D arrangement graphs.

The field of modern graph theory is generally considered to have started with Euler in 1736 with his imaginative solution to the Königsberg Bridge Problem [11]. Graphs are a superb tool for representing information concerning a set of objects and the relationships among the set. They are used extensively in a large number of fields, including mathematics and computer science, in a multitude

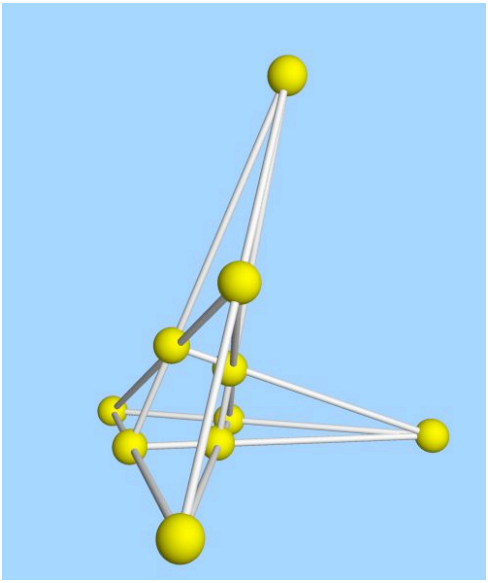

Figure 1.1: A representation of a 5-plane arrangement graph

of different applications to simplify mathematical problems, and to communicate information and relationships.

Because of the interesting mathematical properties graphs exhibit and their use in representing information, there is an ongoing interest in increasing what is known concerning different types and classes of graphs. This thesis undertakes to make a contribution to the field of algorithmic graph theory by investigating the properties of arrangement graphs in 3-space, in order to augment what is already known about arrangement graphs in 2-space. Because it is challenging to visualize and analyse instances of these graphs in 3-space, the tools and strategies used are frequently computational in nature. Hence the thesis has a very computational flavour.

The classes of 3D arrangement graphs consisting of four, five and six planes were systematically investigated and a number of properties, including cardinality up to isomorphism, hamiltonicity and planarity, were determined. The results are summarized in Table 1.

In order to derive the properties of each class of graphs, first the cardinality of the class is established. Each "distinct" model of an arrangement graph, made up of lines of intersection and intersection points, is an instance of the abstract arrangement graph in 3-space. To determine the

Table 1.1: The properties of various classes of arrangement graphs

Property	4-planes	5-planes	6-planes
Cardinality	1	1	43
Hamiltonicity	yes	yes	yes
Planarity	yes	yes	no

cardinality up to isomorphism, all the different ways a new plane P can be added to each distinct instance Q of a graph with $(n-1)$ planes are explored. The different ways are categorized first with regard to the number of edges cut in Q (each cut edge giving rise to a new point) and second, with respect to where the remaining new points of intersection can be located in the space surrounding Q. An instance of every possible graph for n planes in 3-space is produced. For the cases of four and five planes, the number of different possible instances is small and the analysis of these instances can be managed by discussing their geometry in visual terms. For six planes, the number of different instances is just under 250 and requires computational methods to catalogue and analyse each example. Once the cardinality is established, then other basic properties such as Hamiltonicity and planarity are explored.

This research also makes use of a number of results and ideas from another field, computational geometry, which studies arrangements of lines and planes. Although arrangement graphs, as defined in this thesis, have only recently been explored (discussed further in Section 1.2), the arrangement structure from which they are derived has enormous importance in computational geometry (also discussed in Section 1.2). The systematic investigations undertaken in this thesis have implications as to how the structure of the arrangement can be viewed. In this regard, this thesis also contributes to the considerable body of research concerning arrangements in computational geometry.

1.2 Prior Research in 2D Arrangement Graphs and Arrangements

As mentioned above, a small but significant body of work exists investigating 2D arrangement graphs. To the best of our knowledge, no studies on the properties of 3D arrangement graphs have

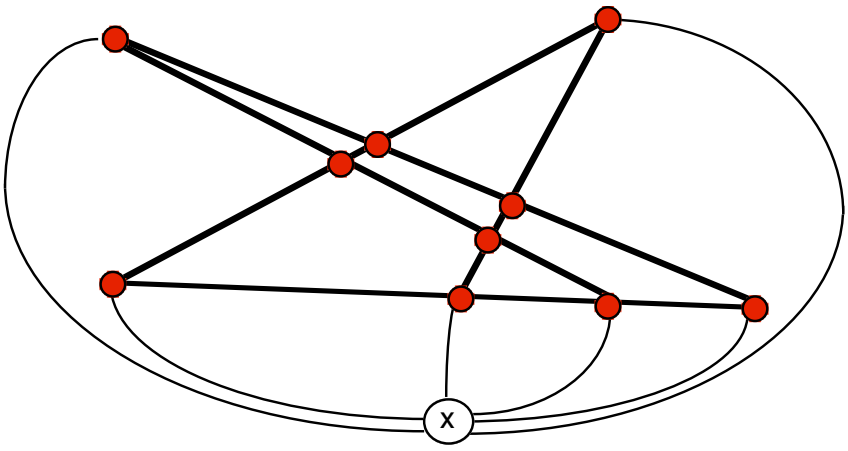

Figure 1.2: G is transformed into a tri-connected graph $G*$

been published. Even though the focus of this study is on 3D arrangement graphs, reviewing the results of the 2D studies can suggest interesting and useful directions for the study of 3D arrangement graphs.

One of the critical problems is identifying whether a given graph G is a 2D arrangement graph or not. Bose et al. [2] refer to this as the *Arrangement Graph Recognition Problem* and establish that it is NP-Hard. This is remarkable given the fact they also show each arrangement graph G has a unique embedding in the plane.

An arbitrary 2D arrangement graph G is planar and can be augmented with an extra vertex x and extra edges joining x to every other vertex that is not degree 4. The effect is to transform G into a tri-connected graph $G*$ as shown in Figure 1.2. Then, using a result from Whitney [21] (which establishes that there is a unique embedding of any planar tri-connected graph in the plane), $G*$ is embedded. Afterwards, x and the extra incident edges on x are deleted leaving the original graph G now embedded in the plane. At this point, it is straightforward to determine the sets of edges incident on each of the vertices. Even so, Bose et al. [2] show that, despite the fact that the unique embedding of a tri-connected graph permits the structure of graph G to be derived, the problem of recognizing whether a given graph is an arrangement graph is still NP-Hard. For this, they turn to the Pseudoline Stretchability problem which states that it is NP-Hard to decide whether the pseudolines

(curves in the plane) of a given arrangement of pseudolines can be "stretched" or straightened out so that all the pseudolines are straight. They reduce the Pseudoline Stretchability problem to the Arrangement Graph Recognition problem.

For 3D arrangement graphs, the analogous problem would be the *3D Arrangement Graph Recognition Problem*, defined as determining whether a given graph is a 3D arrangement graph or not. Unfortunately, there is no graph theoretic result which guarantees a unique embedding for a 3D arrangement graph. Indeed, for a given arbitrary graph, there could be an exponential number of ways that the graph might be represented as a set of planes in 3-space. As a result, it is not clear whether the 3D Arrangement Graph Recognition Problem is even in NP. Hence, the 3D Arrangement Graph Recognition Problem is unresolved at this point and remains an interesting open problem.

Another question is whether 3D arrangement graphs are Hamiltonian. A graph is Hamiltonian if it contains a cycle which traverses all vertices, passing through each vertex exactly once. Bose et al. [2] show that not all 2D arrangement graphs are Hamiltonian with an example of a 7-line 2D arrangement graph which is not Hamiltonian. In this case, the degree-2 vertices are arranged in such a way that a Hamiltonian cycle is impossible. Eu et al. [8] also studied Hamiltonicity in a sub-class of 2D arrangements called sail arrangements. A sail arrangement contains exactly three convex vertices and each line of the 2D arrangement contributes at least one edge to the outside boundary of the arrangement. Sail arrangements are not explicitly defined as arrangement graphs by Eu. et al [8], however the sets of edges and vertices of these sail arrangements are exactly the components of a 2D arrangement graph. Eu et al. establish that a Hamiltonian cycle can always be found for these arrangements.

Another paper by Felsner et al. [9] studies whether Hamiltonian paths and cycles can be constructed for specific classes of arrangement graphs. They examine graphs constructed from pseudocircle arrangements (a family of simple closed curves on the sphere S which have exactly two points in common at which they cross) and projective plane pseudoline arrangements (a family of simple closed curves where every two curves have exactly one point in common). They demonstrate that

pseudo-circle arrangements with $n \geq 3$ circles and projective arrangements with $n \geq 4$ lines are always 4-connected. Then, applying a well-known theorem of Tutte which states that all 4-connected graphs are Hamiltonian, they are able to prove these types of arrangements are Hamiltonian. Felsner et al. [9] also demonstrate that every projective arrangement with an odd number of pseudolines can be decomposed into two separate edge disjoint Hamiltonian paths.

While these results, derived for pseudocircle and projective plane pseudoline arrangements, are interesting, the focus of this thesis is Euclidean 3-space so they are not directly applicable to this study. In this thesis, the three classes of 3D arrangement graphs investigated are all shown to be Hamiltonian but it is still an open question whether all 3D arrangement graphs are Hamiltonian. Nevertheless, a tentative conjecture is made that 3D arrangement graphs are Hamiltonian, and the reasons for this are further discussed in Chapter 5.

Research into arrangements does not always deal with the entire arrangement. One of the components of an arrangement that has received considerable attention in the literature is its envelope, defined as the polygon composed of the bounded edges of all the unbounded faces in the subdivision induced by an arrangement of lines. It is the same as the boundary of an instance w of a 2D arrangement graph. Knowing the structure of the envelope for w is sufficient to compute a number of useful attributes of w such as the diameter, the minimum (or maximum) (x, y) co-ordinates, and the convex hull. Two papers, one by Keil [16] and the other by Eu et al. [8] (mentioned earlier) address various properties of envelopes and possibilities for uncovering envelopes efficiently. In Keil's[16] paper, given a set of n lines in the plane, an algorithm is presented for efficiently constructing the envelope of the arrangement. The portion of the envelope for each of the $2n$ open regions around the outside of the envelope is constructed by making use of the slope information to determine two convex polygonal chains for each open region. The algorithm runs in $O(n)$ time if the slopes of the n indexed lines are sorted, or in $O(n \log n)$ time if the slopes are unsorted and sorting is required as a preliminary step.

Eu et al. [8] also investigate envelopes and their geometric properties, characterizing various envelope polygons in terms of classes of polygons - convex, star-shaped and L-convex. In addition, Eu et al. [8] produce a method, which incorporates Keil's algorithm, for testing whether a given

arbitrary simple polygon of n vertices is induced by an arrangement of lines. A new sub-procedure A is developed which, if the polygon is an envelope, can successfully sort the edges of a polygon in order of orientation in $O(n)$ time. In the full procedure, A is first used to sort the edges of an arbitrary polygon in order of orientation. If the method does not output the edges in sorted order, then the polygon cannot be an envelope. If it does, these edges can be extended to create a set L of sorted lines. Then using Keil's algorithm, the envelope for L is built. If the given polygon is identical to the constructed envelope, then the given polygon must be an envelope polygon. Each method runs in $O(n)$ time so together the two methods only require $O(n)$ time.

In 3D, the corresponding structure to the envelope in 2D is a polyhedron, a region of enclosed space whose boundary is formed by a set of flat faces. Although the idea of developing an envelope polyhedron as a way to study 3D arrangement graphs is an interesting approach, it was not the focus of the study. This could be a rewarding direction for further research into 3D arrangement graphs.

In assessing what background material and tools might be useful for this thesis, it is clear that many of the concepts used to directly study arrangements of lines and planes are also relevant. Arrangements of lines and planes have been investigated extensively for over a hundred years - the classic works by Grünbaum [14], [13] (Chapter 18), and Edelsbrunner [6] provide surveys of past results. Much of ongoing interest in arrangements stems from its position as one of the fundamental structures in computational geometry. Arrangements (of lines, planes and hyperplanes) can be translated into configurations of points and vice versa via a duality transformation so that questions involving a set of points often can be efficiently solved as problems involving sets of lines. Arrangements are also important in motion planning and computer graphics problems. Finally, arrangements are strongly related to two other important structures in computational geometry, Voronoi diagrams and convex hulls. Therefore, arrangements are featured in many classic and contemporary solutions to problems in computational geometry.

Constructing an arrangement is one of the fundamental questions already addressed in the computational geometry literature on arrangements. Edelsbrunner et al. [7] developed the first $O(n^d)$ algorithm for constructing an arrangement of n hyperplanes in d-dimensional space. They used

an incremental construction methodology for inserting an additional plane into an already existing arrangement of planes. This incremental construction algorithm with some modifications still remains very much the method of choice for constructing a complete arrangement. (For recent examples of this algorithm with some modifications, see [4], [20] and [15]). In the incremental algorithm, as each new plane is inserted into an already existing arrangement, only the affected cells of the underlying arrangement are updated. Hence, in 3-space, the algorithm describes how to insert a new plane into an already existing arrangement of planes in $O(n^2)$ time, leading to an $O(n^3)$ algorithm for building the whole arrangement.

The main purpose of the incremental algorithm as presented in[7] is to efficiently build a hierarchical structure of the arrangement. The data structure used to store the arrangement is an incidence graph which records the relationships among the different types of faces, where points are 0-faces, lines 1-faces, faces 2-faces, and polytopes (or convex polyhedrons) 3-faces. Given a category of face, for example k-faces, the incidence graph contains a node for each k-face. Then the relationships of each k-face to all the $(k-1)$-faces and $(k+1)$-faces with which it is incident are also recorded in the incidence graph. However the relationships among the faces at the same level in the hierarchy are not recorded. Fortunately, different data structures have been developed which get around this problem and can be used in connection with this algorithm. Examples are Baumgart's winged-edge structure [20] or Guibas and Stolfi's [20] quad-edge structure.

For this thesis, the ideas behind the incremental algorithm provide two important perspectives. The first is that it is possible to "freeze" the unaffected parts of the arrangement and only update the relatively small part of the arrangement where changes have occurred. The second is that it is possible to analyse the effect of introducing an additional plane into the arrangement. Thus, the algorithm from Edelsbrunner et al. [7] provided the inspiration for regarding an instance of an arrangement graph of n planes as an instance of an underlying arrangement graph of $(n-1)$ planes plus one additional plane.

Another fundamental question which has been addressed in the computational geometry literature is counting the number of different possible arrangements. Edelsbrunner [6] refers to two

arrangements which are the same as combinatorially equivalent. To establish combinatorial equivalence, an analysis of all the different types of cells or k-faces found in the arrangement is required. In the 3D case, the planes induce a division of the space into k-faces where k is $0 \leq k < 3$ and refers to the dimension of the cell. Hence, the number of different arrangements involves analysing 0-faces (points), 1-faces (line segments) and 2-faces (faces). Clearly the components of arrangements studied from this perspective overlap with the components of arrangement graphs (vertices and edges), but include elements that arrangement graphs do not (faces). Nonetheless, it is useful to explore Edelsbrunner's concept of combinatorial equivalence.

Combinatorial equivalence has three different aspects. The first is a one-to-one correspondence between the planes of two arrangements g and h. The second is a one-to-one correspondence between all the k-faces of g and the k-faces of h. The third is a one-to-one correspondence between the position vectors attached to the k-faces in g and h. A position vector specifies the location of a k-face relative to the individual planes of a set H of planes. Again, in the 3D case, each plane in H, denoted by $h_1, h_2, ..., h_n$, creates three sub-spaces:

- the space that is on one side of h_i and denoted by a $+$ sign
- the space that is on the other side of h_i and denoted by a $-$ sign
- the points on the plane itself which are denoted as 0.

Then the relative positions of point p in the space where there are a set H of n planes can be identified by a position vector u with n entries from the set $\{$ +, 0, -$\}$. By convention, the vertical direction is chosen as the "distinguished" one so that points can be characterized as above each h_i (+), below h_i (-) or on h_i (0). The effect of using position vectors in this manner is that two arrangements, exactly the same, except that one has an orientation that is exactly opposite to the other, would not be considered combinatorially equivalent.

Therefore, the concept of combinatorial equivalence requires some modification to be used in the context of a number of possible different *arrangement graphs*. In the field of graph theory, the notion of a distinguished vertical direction used to count the number of graphs is not a conventional one. Nonetheless, Edelsbrunner's approach ([6], p.11) and a considerable body of work in the oriented matroid field ([1], [10]) use position vectors and sign vectors (an alternate name for position vectors) to help establish the combinatorial complexity of different arrangements. For this

thesis, position vectors are not used but the notion behind position vectors inspired the development of a method for segmenting the lines of intersection into different portions. Each portion could be uniquely identified and, as will be shown in later chapters, allowed further methods to be developed for systematically investigating all the graphs which could be created from adding an additional plane to an already existing arrangement graph.

1.3 The Research Goal of this Thesis

The primary goal of this thesis is to establish the structure and properties of classes of 3D arrangement graphs. The purpose of this is to augment the research work already done regarding the properties of 2D arrangement graphs, and to extend what is known about arrangement graphs and arrangements in general. This study took the form of a careful investigation of the simplest classes, arrangements graphs of four, five and six planes.

Three basic properties of these graph classes were researched: cardinality, Hamiltonicity and planarity. Each of these properties is of interest to graph theorists and together, they give an idea of the scope and importance of the class of graphs. In the course of developing solutions to these questions, an in-depth appraisal of the structures in arrangement graphs was conducted. In particular, establishing the cardinality of 3D arrangement graphs composed of six planes entailed developing an extensive methodology to catalogue and evaluate all the possible ways that a set of six planes could interact with each other. In turn, investigating the classes of arrangement graphs also led to a number of insights regarding the complexity of the structures of the underlying arrangements from which wire models of these graphs were derived.

Chapter 2

Terminology

This chapter defines terms which will be used to define and describe arrangement graphs in a rigourous fashion. Some of these terms are already well established in the literature; others are new and defined specifically for this investigation.

It is worth noting that while arrangement graphs, like all graphs, are abstract, many of the properties of these graphs result from the geometry of arranging n planes in general position in $\Re^3$. Hence, it is often useful to refer to the "geometric" instances of the graph, or how a "wire model" of an instance of the graph class exists in $\Re^3$. Therefore, in addition to the terminology for describing the graphs, a number of terms are introduced for describing wire models of instances of these graphs. Since a wire model is a concrete realization of a graph, any properties that we can establish for wire models also directly apply to the associated (abstract) graph.

The definitions are arranged in four categories: terms which refer to arrangements of planes, terms for wire models of graphs, terms for arrangement graphs (once we have the terminology from the first two categories as a foundation) and terms which will be useful for discussing and proving the properties of arrangement graphs.

2.1 Arrangements of Planes and Lines

For the following terms, let $\mathcal{P}$ be a finite set of n planes in general position in $\Re^3$ and let $\mathcal{L}$ be a finite set of k lines in $\Re^2$.

- **arrangement** $\mathcal{A}(\mathcal{P})$ - the connected regions induced by $\mathcal{P}$, in $\Re^3$ with $n \geq 3$ and traditionally denoted as the arrangement $\mathcal{A}(\mathcal{P})$. In $\Re^3$, the regions of $\mathcal{A}(\mathcal{P})$ are composed of cells of 0, 1, 2 and 3 dimensions where a 0-dimensional cell corresponds to a point, a 1-dimensional cell a line, a 2-dimensional cell a face, and a 3-dimensional cell a facet. We are primarily interested in the regions of 0 and 1 dimensions, in other words, the points and lines induced by the planes in $\Re^3$. Furthermore, we insist that the planes be in general position, i.e., each set of three planes must intersect to create a *unique* point, giving rise to exactly $\binom{n}{3}$ intersection points and $\binom{n}{2}$ lines of intersection.

- **i-line** - the line of intersection between two planes in $\mathcal{A}(\mathcal{P})$. Note that the i-line l between two planes P_1 and P_2 appears as a line embedded in each of the planes. In P_1, l is where P_2 cuts through P_1, and in P_2 where P_1 cuts through P_2.

- **arrangement** $\mathcal{A}(\mathcal{L})$ - the connected regions induced by $\mathcal{L}$, in $\Re^2$ with $k \geq 2$ and traditionally denoted as the arrangement $\mathcal{A}(\mathcal{L})$. In contrast to $\mathcal{A}(\mathcal{P})$, the regions of $\mathcal{A}(\mathcal{L})$ are limited to cells of 0, 1 and 2 dimensions. As above, we are interested in the points (0-dimension cell) and lines (1-dimension cell) induced by the lines in $\Re^2$, and similarly require the lines to be in general position, giving rise to exactly $\binom{k}{2}$ intersection points.

- **points of intersection** or **points** - These are defined as the 0-dimensional cells where three planes intersect in $\mathcal{A}(\mathcal{P})$, or, in the 2D case, where two lines intersect in $\mathcal{A}(\mathcal{L})$. We will use the terms *points* and *points of intersection* interchangeably.

2.2 Wire Model

- **partial struct** - Let A and B be two points on an i-line L such that there are no other points on L between A and B. The partial struct is defined as the line segment between A and B.

- **full struct** or **struct** - A full struct is the aggregate of all the partial structs along a specific i-line. It can readily be seen that in $\mathcal{A}(\mathcal{P})$ each full struct contains $(n-3)$ partial structs and $(n-2)$ points, one for each of the $n-2$ planes intersecting the i-line. In 2D, for $\mathcal{A}(\mathcal{L})$, each

full struct contains $(k-2)$ partial structs and $(k-1)$ points. We will use the terms *full struct* and *struct* interchangeably.

- **3D wire model or W3D**$(\mathcal{P})$ - consists of points and line segments induced by $\mathcal{A}(\mathcal{P})$ as follows:
  - the points in W3D$(\mathcal{P})$ correspond to the points in $\mathcal{A}(\mathcal{P})$.
  - the line segments W3D$(\mathcal{P})$ are partial structs as defined earlier in this section.
- **2D wire model or W2D**$(\mathcal{L})$ - consists of points and line segments induced by $\mathcal{A}(\mathcal{L})$ as follows:
  - the points in W2D$(\mathcal{L})$ correspond to the points in $\mathcal{A}(\mathcal{L})$.
  - the line segments W2D$(\mathcal{L})$ are partial structs as defined earlier in this section.
- **extreme point on a struct or an i-line** - the point dividing the portion of the i-line that is a struct, and therefore part of $W3D(\mathcal{P})$ or $W2D(\mathcal{L})$, and the portion of the i-line that is not part of $W3D(\mathcal{P})$ or $W2D(\mathcal{L})$. There is an extreme point at each end of each struct.

2.3 Arrangement Graphs

- **3D arrangement graph** - denoted as $G(\mathcal{P})$ and defined as the graph whose vertex set corresponds to the points in W3D$(\mathcal{P})$ and whose edges correspond to the partial structs in W3D$(\mathcal{P})$.
- **2D arrangement graph** - denoted as $H(\mathcal{L})$ and defined as the graph whose vertex set corresponds to the points in W2D$(\mathcal{L})$ and whose edges correspond to the partial structs in W2D$(\mathcal{L})$.
- **classes of 3D arrangement graphs** - The set of graphs created by a set of n planes is denoted by $AG3D_n$.
- **classes of 2D arrangement graphs** - The set of graphs created by a set of k lines is denoted by $AG2D_k$.

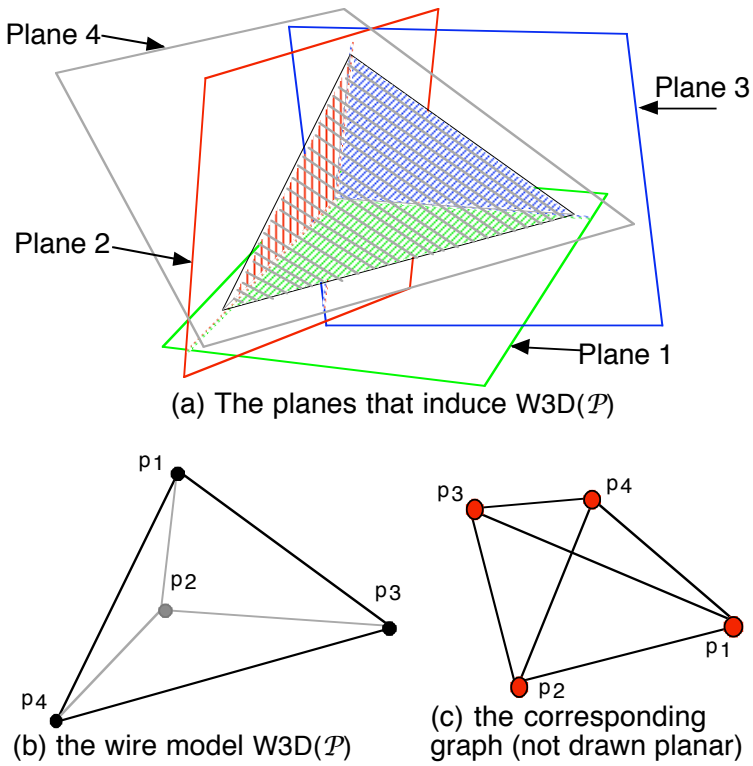

Figure 2.1: A set of planes $\mathcal{P}$ inducing a W3D($\mathcal{P}$) and the corresponding graph

Figure 2.1 shows an example of how a set of planes induces a wire model of an arrangement graph. Note that the graph can be drawn in a multitude of different ways, some of which appear very different from the underlying arrangement of planes.

2.4 Additional Terminology for Analysing the Classes of $AG3D_n$ and $AG2D_n$

The first two terms below define the two possible ways that i-line L and the struct on L can be traversed; the last two terms precisely describe specific parts and points of l. For these definitions, let A and B be the two extreme points on i-line L and hence the end points of the struct.

- **Direction beyond the extreme point** - In relation to an extreme point A on a struct of an i-line L, the *Direction beyond the extreme point* is the direction towards that part of L which is not part of the struct. (Figure 2.2)

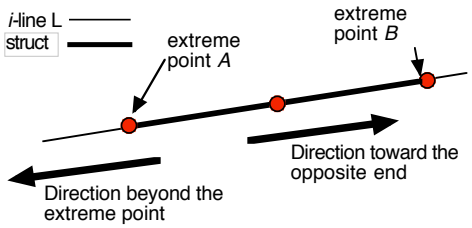

Figure 2.2: Direction in relation to an extreme point A

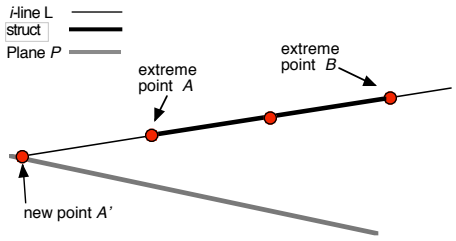

Figure 2.3: I-line L intersects with P beyond A

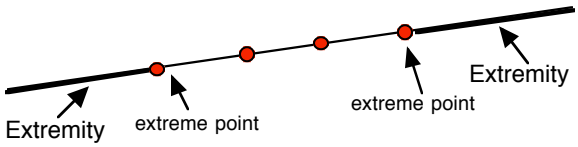

Figure 2.4: The extremities of i-line L

- **Direction toward the opposite end of the struct** - For the extreme point A on a struct of L, this defines the direction which traverses the struct to reach the other extreme point B at the opposite end of the struct. (Figure 2.2)

When a new plane P_i is added to a wire model m, new intersections occur in two ways — either by P_i cutting partial structs of m or where P_i cuts the i-lines of m beyond the ends of their extreme points.

- **Intersection beyond an extreme point** - This refers to the intersection A' of a plane P_i and i-line L which occurs beyond the extreme point A on L. (Figure 2.3)

- **i-line extremity** - For an i-line L which contains a struct, an extremity is the ray from one of its extreme points to infinity. Figure 2.4 shows that each i-line L has two extremities: one

is the portion of L beyond one of its extreme points, the other is the portion of L beyond the other extreme point. See Figure 2.4.

For deriving the number of possible graphs for $AG3D_6$, a number of definitions for determining whether graphs are equivalent to each other were used. In the following discussion, two graphs G_1 and G_2 will be used to denote two graphs with sets of vertices V_1 and V_2 respectively.

- **Graph isomorphism** - Two graphs G_1 and G_2 are isomorphic if and only if there is a function f from V_1 onto V_2 where uv is an edge joining two vertices u and v in G_1 if and only if there is an edge $f(u)f(v)$ joining two vertices $f(u)$ and $f(v)$ in G_2.

If two graphs are isomorphic, it is possible to rename the vertices of one graph with the labels of the vertices in the other. Everywhere there is an edge in one graph, there will be a corresponding edge in the other graph. Therefore, it is clear that the two graphs have the same structure.

- **Equivalence class of graphs** - When all graphs that are isomorphic to each other are grouped together in one class, they are said to form an equivalence class of graphs.

The next definition is used in connection with deciding whether two *arrangement* graphs are equivalent to each other. A new arrangement graph of n planes will be created incrementally by adding a plane P to an underlying wire model Q of an arrangement graph with $(n-1)$ planes. A number of new points are created, one on each i-line in the underlying wire model, and one portion of each i-line is intersected, either a partial struct or an extremity. Obviously there are an infinite number of ways P can be added to Q. However, if two planes P_1 and P_2 cut exactly the same list of partial structs and i-line extremities in Q, then it is clear that the two new wire models created have the same adjacency lists and the graphs induced by these wire models are isomorphic to each other. When considering the number of possible arrangement graphs, therefore, only one example of the wire models produced from a set of planes R which cut the same list of partial structs and extremities needs to be considered. All the planes in R may have different orientations but, in terms of inducing arrangement graphs, they are all equivalent. This motivates the following definition which is used to decide which sets of planes are equivalent.

- **Equivalence class of planes** - All sets R of n planes each which cut exactly the same list of partial structs and extremities of the i-lines in an underlying wire model Q of an arrangement graph forms an equivalence class of planes.

Chapter 3

General Properties of $AG3D_n$

First, two properties of all classes of 3D arrangement graphs are presented which can be readily derived from the definitions in the terminology section. After this, a number of other properties are listed which will be used in discussions and proofs throughout the thesis. Finally, the two simplest classes of 3D arrangement graphs - $AG3D_4$ and $AG3D_5$ - are examined. These two classes provide a number of insights into the general form and structure of arrangement graphs and help provide a good foundation for studying other classes of graphs, in particular $AG3D_6$.

3.1 General Properties of 3D Arrangement Graphs

Let $\mathcal{P}$ be a set of n planes in general position.

Property 1. *The number of vertices in a 3D arrangement graph is* $\binom{n}{3}$.

There is a vertex for each point in an associated wire model. From the definition of the arrangement $\mathcal{A}(\mathcal{P})$, each set of three planes intersects to create a unique point so there are $\binom{n}{3}$ points, and hence the same number of vertices.

Property 2. *The number of edges in a 3D arrangement graph is*

$$n \cdot (n-1) \cdot (n-3)/2.$$

Each pair of planes in $\mathcal{A}(\mathcal{P})$ creates a line of intersection and thus there are $n \cdot (n-1)/2$ i-lines. For any given i-line l in a wire model of an arrangement graph, all the other $n-2$ planes

intersect l. Thus, on l, there are $(n-2)$ points of intersection and $(n-3)$ partial structs (the number of intervening line segments between the points on l). Therefore in any wire model, there are $n \cdot (n-1) \cdot (n-3)/2$ partial structs. Hence the graph has the same number of edges.

Property 3. *If one plane is removed from the wire model of a graph in $AG3D_n$, the result is a wire model of a graph in $AG3D_{n-1}$.*

Observation 1. *Each plane of a wire model of a graph in $AG3D_n$ contains a wire model of a graph in $AG2D_{n-1}$.*

Each plane of a wire model in $AG3D_n$ must contain $(n-1)$ lines in general position. Hence, the conditions required for creating a wire model of a graph in $AG2D_{n-1}$ are met. This result is also stated without proof in [6] p. 93.

Property 4. *The degree of each vertex of a graph in $AG3D_n$ must be one of 3, 4, 5 or 6.*

This follows from the construction of an arrangement graph. If the point in the associated wire model is the extreme point on all three i-lines incident on the point, the point will be degree 3. If it is the extreme point on two i-lines, it will be degree 4, if extreme on one i-line, it will be degree 5, and if extreme on none, it will be degree 6.

Property 5. *Each struct in a wire model of a graph in $AG3D_n$ is contained in exactly two planes of the wire model.*

Property 6. *Each point in a wire model of a graph in $AG3D_n$ is contained in three planes of the wire model.*

3.2 4-Plane Arrangement Graphs ($AG3D_4$)

Three key properties of the class $AG3D_4$ can be readily determined.

Lemma 1. *The cardinality of $AG3D_4$ is one.*

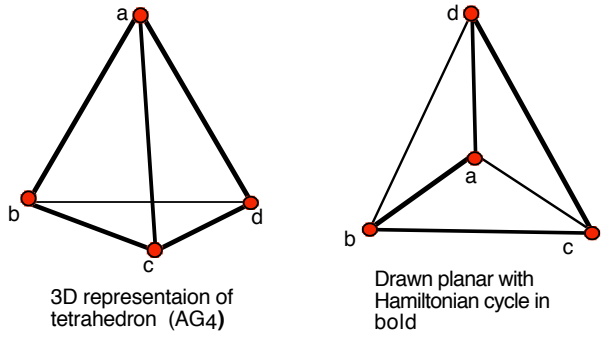

Figure 3.1: A drawing of the graph s in $AG3D_4$

Proof. Consider a wire model of any four planes. These planes create four points and each of these four points is connected to the remaining three points. Therefore there is only one wire model possible which is a tetrahedron where each point is degree 3. See Figure 3.1. Hence it follows that there is only one graph σ in $AG3D_4$. □

In the rest of this study, the single graph in $AG3D_4$ will be denoted as σ and the wire model of σ will be denoted as s. Next, the properties of σ are examined.

Lemma 2. *The graph σ is planar and Hamiltonian.*

Proof. Figure 3.1 shows that σ is planar and Hamiltonian. □

Corollary 1. *The only wire model (and the only graph) in $AG2D_3$ is a triangle.*

A triangle is the only possible graph for three lines arranged in a plane so that each line intersects the other two.

3.3 5-Plane Arrangement Graphs ($AG3D_5$)

The results listed below follow directly from Properties 1 and 2 and the definition of an arrangement $\mathcal{A}(\mathcal{P})$.

- Each graph in $AG3D_5$ has 10 vertices, six more than for the graph in $AG3D_4$, and 20 edges, 14 more than in $AG3D_4$.
- In a wire model r of a graph in $AG3D_5$, there are three points and two partial structs on each i-line.
- There are 10 i-lines in r.

To study $AG3D_5$, another plane is added to the tetrahedron wire model of $AG3D_4$ and all the different ways that the new plane can interact with the tetrahedron are enumerated. To assist in this analysis, first two useful lemmas are presented. The first, Lemma 3, considers the case in 2D of adding another line in general position to the only graph in $AG2D_3$ (a triangle).

Lemma 3. *The cardinality up to isomorphism of the class of 2D arrangement graphs $AG2D_4$ is one and the graph always has one degree-4 vertex.*

Proof. From Corollary 1, the only graph for $AG2D_3$ is a triangle. If another line is added, there are two cases to consider. The first is that a new i-line l does not intersect any of the partial structs in the triangle. The second is that l does intersect the triangle.

In the first case, l cannot be parallel to any of the other i-lines in the triangle. Without loss of generality, let the i-lines containing structs $struct_b$ and $struct_c$ in Figure 3.2 intersect with l beyond p_2 and p_1 respectively. The i-line containing $struct_a$ must intersect with l beyond either p_1 or p_2. Again, without loss of generality, let the i-line containing $struct_a$ intersect with l beyond p_1. Three new points are created. Point p_1 is degree 4, p_2 is degree 3, and p_1 remains degree 2. Since the three i-lines in the plane of s form a triangle, no matter how l is introduced, the result will be the same as in Figure 3.2.

In the second case, l slices through a triangle in the plane, and must cut exactly two out of the three partial structs. The third struct is extended to intersect with l. The resulting wire model, shown in Figure 3.3, is clearly isomorphic to the wire model in Figure 3.2 .

Therefore the cardinality of $AG2D_4$ is one and the resulting 2D graph always has one degree-4 vertex. □

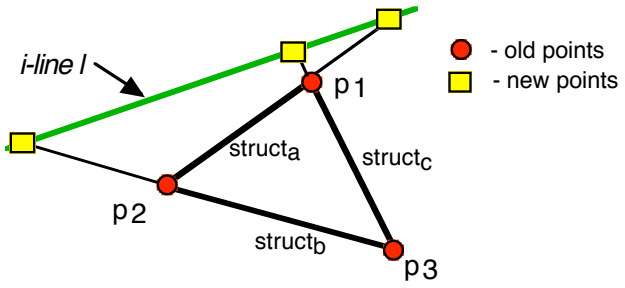

Figure 3.2: A wire model of the graph in $AG2D_4$ when the structs of $AG2D_3$ are not cut

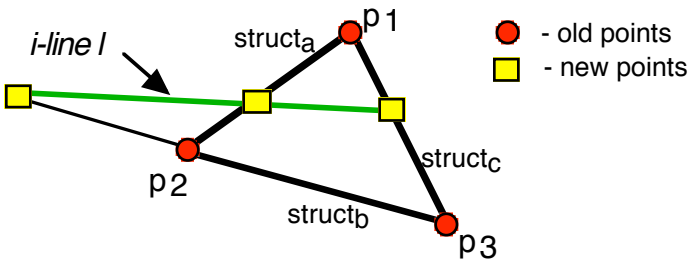

Figure 3.3: A wire model of the single graph in $AG2D_4$ when the structs of $AG2D_3$ are cut

In the rest of this study, the unique graph in $AG2D_4$ will be designated as ω and the wire model of this graph will be referred to as w. The next lemma considers the effect on the wire model s of the unique graph in $AG3D_4$ when an additional plane P_5 introduced so that it does not cut any of the partial structs of s. Plane P_5 must be in general position to the set of planes involved in s in order to form an arrangement of five planes.

Lemma 4. *If a fifth plane P_5 is introduced in general position so that no structs in the wire model s of the graph in $AG3D_4$ are cut, exactly one of the degree-3 points in s is transformed into a degree-6 point.*

Proof. Allow the plane P_5 to move towards s until it comes into contact with s. Since P_5 is in general position, it cannot contact the three points in one of the planes of s simultaneously because this would imply that P_5 is parallel to one of the planes of s. Similarily, it cannot contact two points

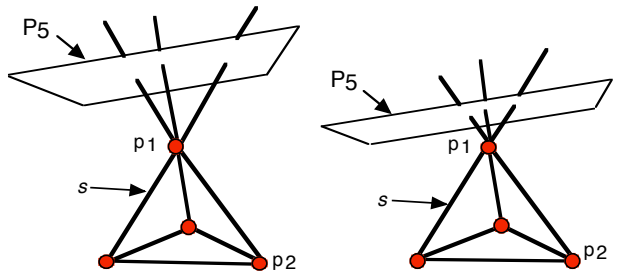

Figure 3.4: The orientation of P_5 determines how it intersects with s

simultaneously since this would imply that P_5 is parallel to one of the i-lines of s. As a result, P_5 must initially contact a single point p_1 in s. Imagine then that P_5 is backed off slightly so that it breaks contact with s. Clearly the three i-lines incident on p_1 must intersect with P_5 beyond p_1.

From the construction of the tetrahedron s, note that each of the remaining three points in s shares a specific i-line with point p_1. Because P_5 in general position with respect to the original planes in the tetrahedron, it follows that, without loss of generality, any other point p_2 in s can have at most two of its i-lines intersect with P_5 beyond p_2. Therefore, any other point in s at most can be transformed into a degree-5 point. Note that p_1 becomes a degree-6 point as shown in Figure 3.4. □

This lemma establishes that, if P_5 does not intersect s, it is the orientation of P_5 with respect to s, not the distance of P_5 from s, that determines how the i-lines of s will intersect with P_5. This result is useful because it determines how one of the points in s is affected by the introduction of a new plane P_5. Using this result, other implications of introducing P_5 can be uncovered.

Theorem 1. *The cardinality of AG3D$_5$ is one.*

Proof. As in Lemma 3, there are two cases. First, a fifth plane P_5 is introduced into a wire model s of σ so that no structs of s are cut. In the second case, the original structs of s are cut.

No structs of s are cut.

From Lemma 4, three of the six structs in s must be extended beyond one of the points in s, turning that point into a degree-6 point. Without loss of generality, assume that this point is p_1 in

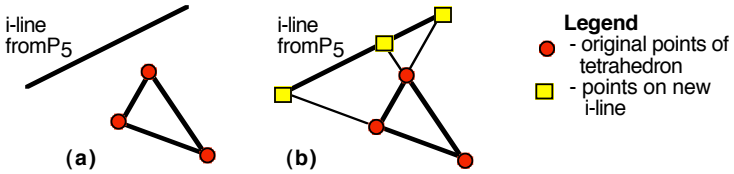

Figure 3.5: Determining the three new points in the P_4 plane

Figure 3.4. The fourth plane P_4 is shown in Figure 3.5(a) and does not contain p_1. Then, three new points are created in P_4 by extending the existing structs in P_4 to intersect with the new i-line resulting from P_5. By Lemma 3, there is only graph in $AG2D_4$ so the points in P_4 must be arranged as shown in Figure 3.5(b).

Now consider the way the points in the new plane P_5 are configured. In Figure 3.6, the three new points connected to p_1 form a triangle in P_5 which is a scalar copy of the one in P_4 but inverted. The other three new points in P_5 form a straight line and *are the same three points as in* P_4 because this line is the line of intersection between P_4 and P_5. Furthermore, since the points in P_5 are a wire model of the only graph in $AG2D_4$, they also form the same pattern as in Figure 3.5(b). The two planes are attached as shown in Figure 3.6. Again because the tetrahedron is symmetric, no matter which point p_1 is chosen as the degree-6 starting point, the relationship of the points in the new plane P_5 to the points in the plane of s which does not contain p_1 is always the same. Figure 3.7 is augmented by adding point p_1 (now the degree-6 point) and the last three structs are added. This result shows there is only one wire model possible when P_5 is introduced so that none of the structs of the tetrahedron are cut.

The structs of s are cut.

For the second case P_5 is allowed to slice through s in two ways, first by cutting three i-lines in s and then four i-lines in s. First, assume that P_5 cuts three i-lines. Then it can be easily seen that P_5 produces a smaller tetrahedron within the larger tetrahedron. Thus this case can be reinterpreted as the four planes of the smaller tetrahedron being the "original" planes and the remaining plane as the "fifth" plane introduced into the arrangement. Therefore, this situation is equivalent to the case

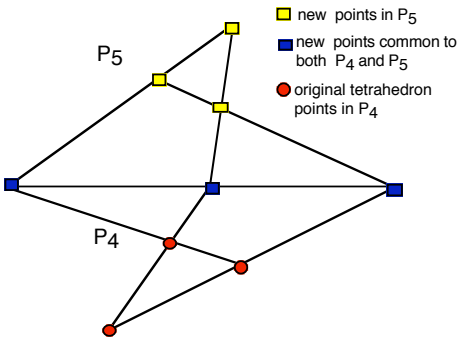

Figure 3.6: The relationships of the points in planes P_4 and P_5

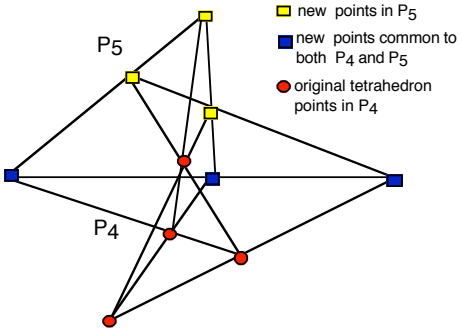

Figure 3.7: Only one possibility for the points in P_5 with regard to s

above where no structs are cut.

Finally, let P_5 slice through s cutting four structs of s. Then three of the four original planes and P_5 can be selected to form a tetrahedron outside of the original s. Once again, the four planes creating this new tetrahedron can be considered to be the "original" ones. The last plane, which does not cut any of the structs in this newly formed tetrahedron, can be relabelled as the "fifth" plane. Therefore this case is also equivalent to the case where no structs are cut.

Thus, this proves there is only one wire model of a graph in $AG3D_5$ possible and that the cardinality of $AG3D_5$ is one. □

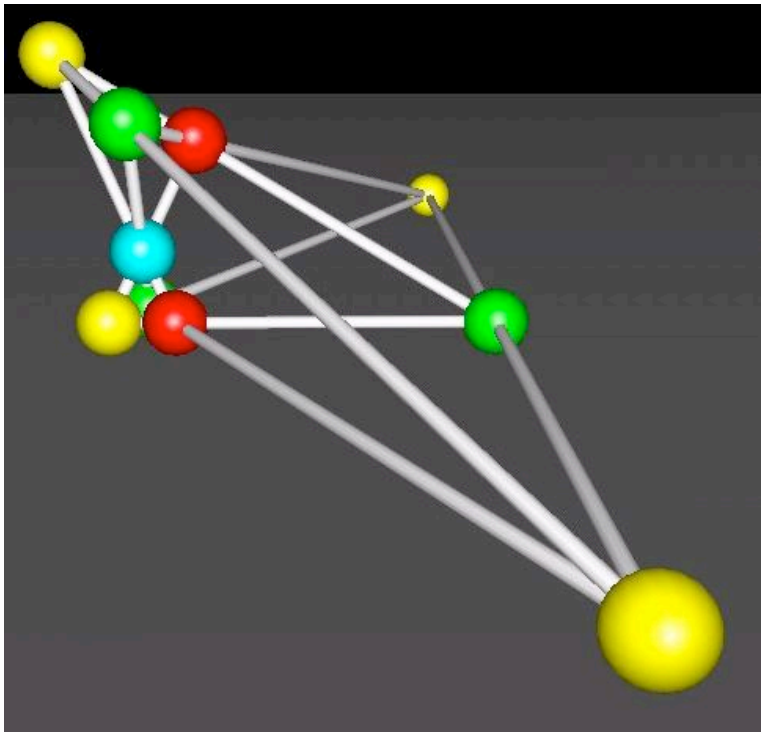

Figure 3.8: The wire model r of the single graph κ in $AG3D_5$

In the rest of this study, the one graph in $AG3D_5$ will be designated as κ and the wire model of this graph will be referred to as r. The wire model r is shown in Figure 3.8. Because there is only one graph in $AG3D_5$, the five planes, the 10 points and 10 i-lines of $AG3D_5$ can be assigned individual labels. The labelling adopted is shown in Figure 3.9 and will be used throughout the rest of the thesis.

Theorem 2. *$AG3D_5$ is planar.*

Proof. The graph drawing in Figure 3.10 demonstrates that the single graph in $AG3D_5$ is planar. □

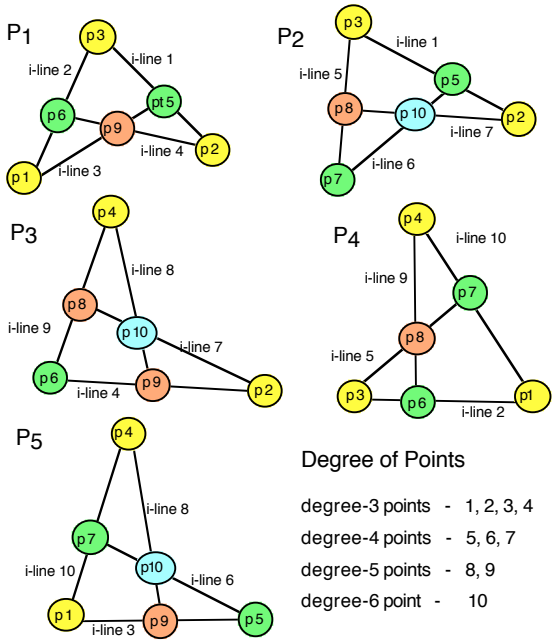

Figure 3.9: The five planes of r

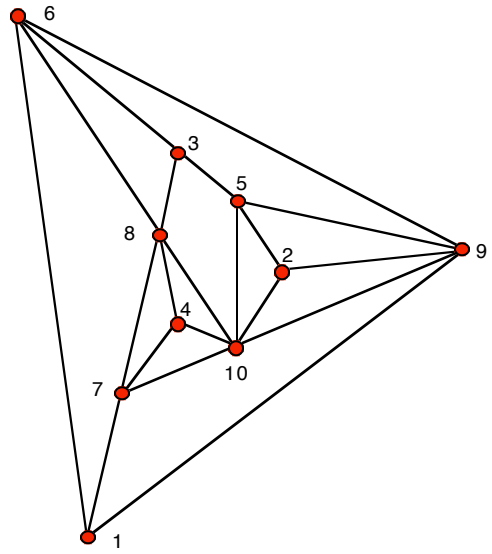

Figure 3.10: The single graph in $AG3D_5$ drawn planar

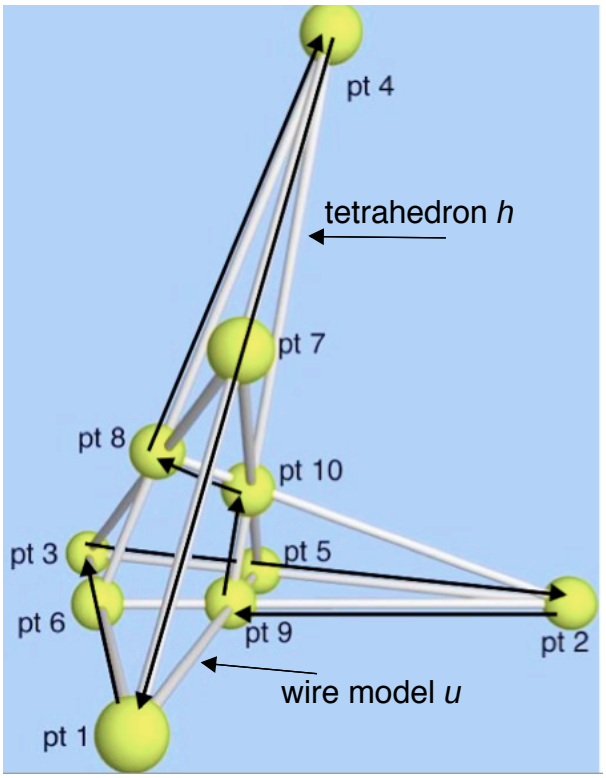

Figure 3.11: The single graph in $AG3D_5$ is Hamiltonian

Theorem 3. *$AG3D_5$ is Hamiltonian.*

Proof. The wire model r of the single graph in $AG3D_5$ in Figure 3.11 can be viewed as a tetrahedron h, consisting of points p_4, p_7, p_8 and p_{10}, attached to a wire model u of a single graph in $AG2D_4$, consisting of points p_1, p_6, p_3, p_5, p_2 and p_9. In r:

- one point in h is attached to three points forming one of the two triangles in u
- one point in h is attached to two points in the quadrilateral in u
- one point in h is attached to one point in the second triangle in u
- the final point in h is not attached to u

One of the Hamiltonian cycles in r is traced out in Figure 3.11. □

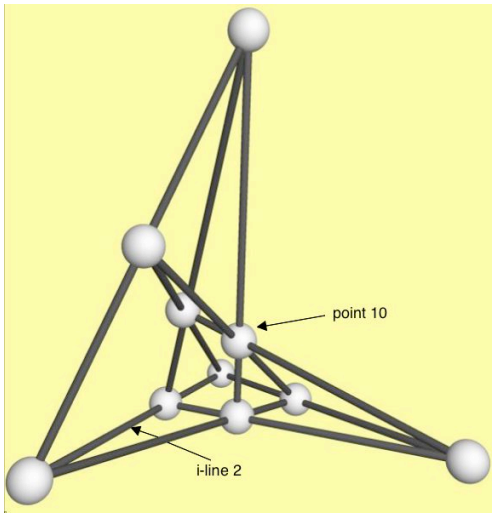

Figure 3.12: The wire model r is symmetric

3.3.1 Types of degree-3 points in r

There are two types of degree-3 points in any wire model r. Points p_2 and p_4 are always points on a triangle in the planes where they are found, but p_1 and p_3 are not. For example, p_1 in Figure 3.9 is a point on a quadrilateral in planes P_4 and P_5 but a point on a triangle in P_1. These differences will have implications in the study of $AG3D_6$.

Definition: - A **degree-3-tri point** is contained in a triangular face in each plane of $AG3D_5$, whereas a **degree-3-quad point** is contained in a quadrilateral face in at least one plane of $AG3D_5$.

Property 7. *$AG3D_5$ is symmetric about a plane through i-line$_2$ and p_{10}.*

Proof. This can be established from the construction of r and can clearly be seen in Figure 3.12.

Chapter 4

Properties of $AG3D_6$

4.1 Background to Studying the Graphs in $AG3D_6$

An incremental approach is again used and all the different ways that a sixth plane can be added to the wire model of the single graph in $AG3D_5$ are investigated. Recall the notation introduced earlier:

- For $AG3D_5$, the single graph is denoted as κ and the corresponding wire model as r.
- For $AG3D_4$, the single graph is denoted as σ and the corresponding wire model (a tetrahedron) as s.
- For $AG2D_4$, the single graph is denoted as ω and the corresponding wire model as w.

Finally, u will denote an arbitrary graph in $AG3D_6$.

To create u, imagine a sixth plane P_6 is added to r. The discussion below speaks of P_6 as "interacting" with r. This means that all the i-lines in r either intersect with P_6 within the structs of r or in the i-lines' extremities. Even if P_6 completely misses r, it will still "interact" with r in that the ten structs of r must be extended along their respective i-lines to the points where they intersect with P_6. Similarily, in each of the planes of r, P_6 interacts with the 2D wire model w. Here, the image of P_6 is an additional i-line which intersects with each of the other i-lines contained in the plane.

The intuitive approach is to count all possible ways of adding a sixth plane P_6 in general position to r. The obvious difficulty with this approach is how to write an algorithm to accomplish it. The

methodology selected for this study is to investigate the effect of a sixth plane on r, and for this, it is clear that a sixth plane must intersect with each of the i-lines in r. Each i-line has four different *portions*, two partial structs and two extremities. Therefore, each distinct equivalence class of a sixth plane must interact with exactly 10 portions of the i-lines of r, one portion per i-line. Since r has 40 portions, a brute force algorithm would consider $\binom{40}{10}$ subsets of portions and determine which ones represent valid interactions with a sixth plane.

However, by considering the structure of r, many invalid subsets can be eliminated. For example, two partial structs from the same struct m cannot both be included in the same list of cut partial structs as this would imply the sixth plane intersects m twice. This is prohibited if the sixth plane is in general position. Where it is clear that the intersections created by the cut partial structs do not allow a straight line in a particular plane, then it is also clear that it would not permit a valid plane in the 3D arrangement of planes. If the location of the points does allow a new line in a plane of r, it is said to *support* a new line, or if there are 10 new points which allow a new plane in 3D, then the 10 points are said to *support* a new plane. After filtering all the possible sets of i-line portions, 245 possibilities remain, and the associated graphs are generated. Some of these resulting graphs may not be valid arrangement graphs if some of the distinct sets of 10 new intersection points added to these graphs do not support a plane.

Nonetheless, all 245 graphs are processed using a graph isomorphism software package called Nauty [18] which determines which are isomorphic. Nauty determines there are 43 equivalence classes of graphs.

Finally, each of the 43 graphs is checked to ensure it is a valid arrangement graph, and indeed all are arrangements graphs. They are presented in Appendix B. The graphs and a wire model for each are also provided on a web-site `(www.cs.uleth.ca/~nickle/arrangements)` in a format suitable for computational processing.

Even though an incremental approach is used to study $AG3D_6$, the procedures used to create and analyse the graphs in $AG3D_6$ are very different from those used to study the wire model r. This is because there are more ways for a new plane P_6 to interact with r. When a fifth plane P_5 is added to s, there are only three possible ways that P_5 can interact with s, but when P_6 is added to r, it can

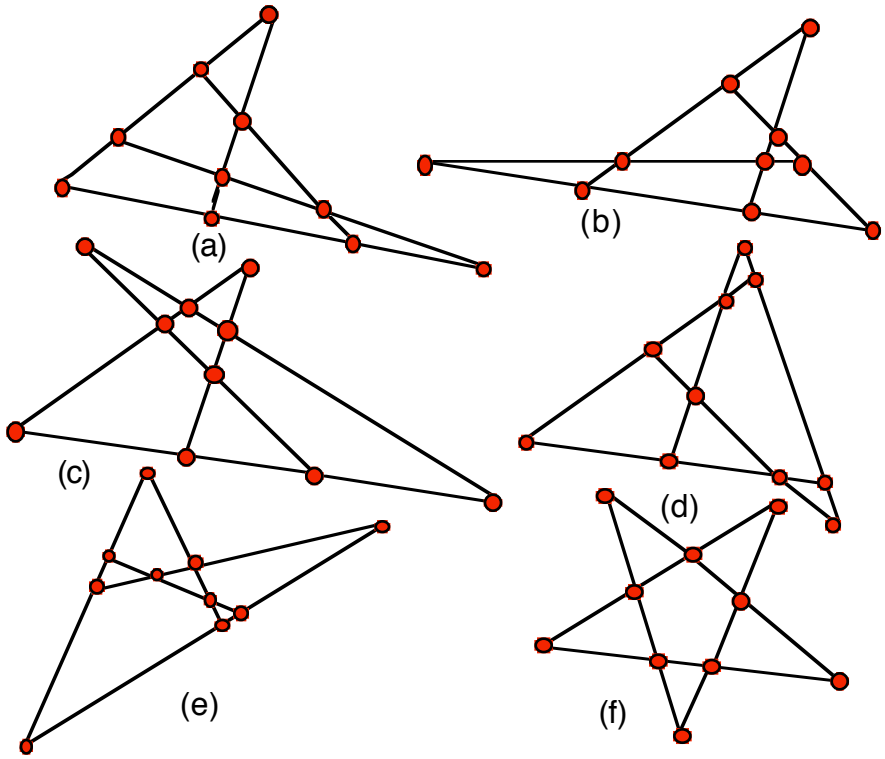

Figure 4.1: Wire models of the six graphs in $AG2D_5$

completely miss r or it can cut a variety of numbers of partial structs in r. Section 4.2.4 shows that P_6 can cut either 3, 5, 6, 7, 8 or 9 partial structs of r and that, in all, there are 33 distinct ways that P_6 can interact with r.

Secondly, recall that no matter how P_5 interacts with a plane of s, the same wire model w is always produced in each plane of s. On the contrary, when P_6 interacts with w in a plane of r there are six possible outcomes. A set of wire models for the six graphs in $AG2D_5$ is shown in Figure 4.1. As a result, it is clear that there will be considerably more variety in the class of graphs in $AG3D_6$ than there will be in the class of $AG3D_5$.

4.1.1 Methods for constructing arrangements

To build the arrangements for this study, the algorithm presented in Edelsbrunner et al.[7] was considered. However, it became apparent that this algorithm did not meet the specific requirements

of this study. It is well suited to efficiently building the data structures to store an arrangement, but only *after* all the intersections of a set of planes have been specified. However, uncovering all the different ways that a set of planes can intersect with each other is the main challenge of this study. So it would be necessary to modify Edelsbrunner's approach. Secondly, as mentioned previously in Chapter 1.2, the incidence graph, which stores the arrangement in Edelsbrunner's algorithm, does not directly store adjacency list information. Given these considerations, specialized data structures and methodologies to fit the needs of this study were developed, while at the same time still retaining the incremental-plane approach.

4.2 The Methodology - Five Main Steps

A high-level overview of the algorithm is that, for Steps One through Four, it uncovers distinct instances of sets of 10 new points placed in a fashion that is consistent with the points supporting a plane. Then in Step 5 a wire model of the graph is constructed and co-ordinates for the points of the wire model are generated. Note that, until this final check is done, there is no guarantee that the points actually support a plane. For each equivalence class of (possible) graphs in $AG3D_6$, the 10 new points do support a plane but it is necessary to do the fifth step to prove this conclusively. The algorithm is useful because it eliminates all the invalid sets of point locations and all graphs which are isomorphic to each other. In the final analysis, only 43 cases need to be checked out of a theoretical possible number of $\binom{40}{10}$.

The first two steps address the task of generating a list of all the outcomes of the different ways a sixth plane P_6 could intersect sets of partial structs in r. Note that, at this stage, the points created in the extremities of the i-lines are not considered. The output of Step Two is all the valid ways that P_6 can slice through r. In terms of equivalence classes, only part of the information to identify each equivalence class is available at this point, namely which partial structs in r are cut.

The third step focusses on procedures to identify all the possible ways the remaining new points in P_6 can be placed in the extremities of the existing i-lines. Each time a set of 10 positions for the new intersections is identified that is consistent with the points supporting a plane is found,

the graph is output. The fourth step prunes the list of graphs to eliminate the graphs which are isomorphic to each other. Finally, the fifth step checks an example of each equivalence class of graphs to determine whether an actual wire model arrangement of six planes can be constructed for that equivalence class.

4.2.1 Step One - Generates all the ways the points in r can be partitioned into two subsets

The task of determining the effect on a sixth plane P_6 on r is supported by the following observation:

Observation 2. *A new plane P_6, inserted into r, partitions all the points in r into two subsets.*

Each equivalence class of a sixth plane intersecting with r produces different subsets of points. Hence, to examine all the different ways a sixth plane can intersect r, it is necessary to consider all the ways the points in r can be partitioned into two subsets. For this, the set of indices C, corresponding to the ten points in r, are split into two subsets, A and A', where A' is the complement of A. Clearly, dividing C into two by isolating the points of A is the same as dividing C by isolating the points of A'. Therefore, it is not necessary to generate and store the corresponding A' for each distinct A. As a result, the procedure below only outputs distinct instances of A. Notice that the two subsets, C and $\emptyset$, are a pair of valid subsets and represent the case where P_6 completely misses r. For this situation, the set C will be stored. The procedure in Step One is as follows:

Input: the set of indices C corresponding to the ten points in r. The set of indices adopted are the labels previously assigned to r (Figure 4.3).

Procedure:

Generate and output all the combinations of the indices of C.

Output: the power set of C.

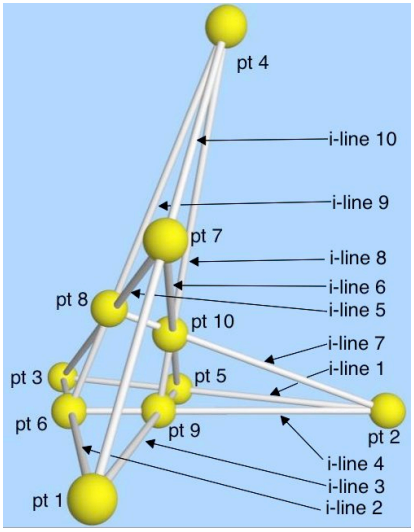

Figure 4.2: The wire model r

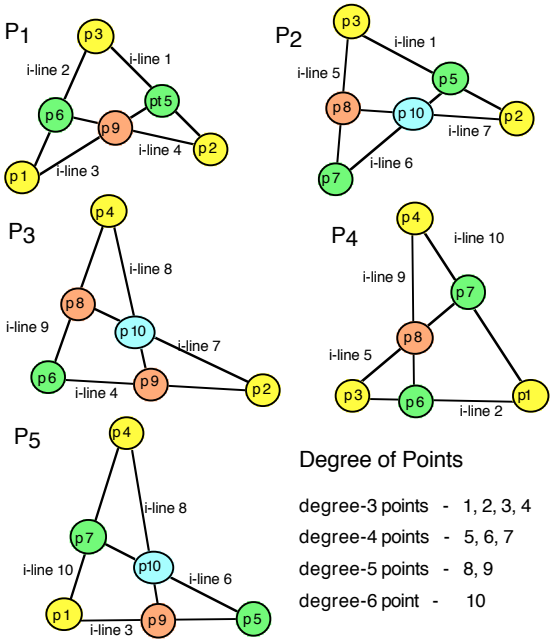

Figure 4.3: The labels on the five planes $P_1, P_2, ...P_5$ of r

4.2.2 Step Two - Generates all the *valid* ways a new plane P_6 can partition the points of r into two subsets

In Step Two, the first goal is to transform the output from Step One into a list of partial structs cut. Then, the second goal is to eliminate the lists of partial structs which do not meet the necessary criteria (defined below) for a valid intersection of a new plane P_6 with r. For this step, two lemmas are required.

Lemma 5. *A new plane P_6 inserted in general position into r cannot cut the same struct twice.*

Proof. If P_6 intersects two different partial structs from the same struct, it clearly cannot be in general position because general position requires that each plane intersects with every other plane exactly once. □

When a new plane P_6 cuts r, a set of new i-lines $l_1, l_2, \ldots, l_5$ are created, one i-line per plane $P_1, P_2, \ldots, P_5$ respectively. Each new line l_i represents the line of intersection between P_6 and P_i. The next lemma considers the implications of the fact the line of intersection is straight.

Lemma 6. *If plane P_6 cuts a face f in the wire model w of the single graph ω in $AG2D_4$, it cuts the boundary of f in exactly two places.*

Proof. Each of the faces f produced in w is convex meaning that, for each pair of distinct points a, b in f, the line segment with endpoints a and b is also contained in f. Therefore, if three (or more) partial structs of f are cut, it is impossible to join the cut points on these partial structs with a straight line. However, for the new line l to represent the image of plane P_6 in w, it must be possible to represent it as straight. Similarly if a face f of w has only one partial struct cut, this implies that the new line l is a ray (a line that begins at a certain point and extends forever in *one* direction only). If l is a ray, it cannot represent P_6 in the plane P_x. Therefore the line l representing P_6 cuts each face f in exactly two places. □

While the power set of C consists of lists of points from r, the discussion above refers to the list of cut partial structs, not lists of points. Note that each point in r is attached to at most six points

because, from Property 4, no point can have degree greater than six. For this study, a data structure X was developed (to be described briefly in Section 4.3.1 and in detail in Section 4.3.8) which stores the complete list of partial structs for r. Therefore, it is not difficult to identify the partial structs associated with each point and hence relatively simple to transform the lists from points to partial structs. The procedure for Step Two is as follows:

Input: the power set of C and the list of structs and partial structs in r.

Procedure:
Let the list E be a set of partial structs from r.

for all c_i **do**
  A new list E is initialized to the empty set.

  for each index k in c_i **do**
    All the partial structs attached to the point p_k are added to E.
    Any partial struct which appears twice in E is eliminated.

    if E does not contain two partial structs from the same struct AND each face in E contains exactly two partial structs **then**
      E is recorded as an element s_i of the set S of the valid lists of partial structs.
    end if
  end for
end for

Output: a set S of lists of partial structs in r, with each element s_i of S representing a *valid* way that P_6 can cut the structs of r.

4.2.3 Correctness of procedures in Step Two

First, the reason for eliminating any partial struct which appears twice in the list E is explained. The only way that the same partial struct can be selected twice for E is if both ends of the partial struct are included in the list of points c_i. But if both endpoints are in c_i, then clearly the new plane P_6 does not pass between these two points and this partial struct cannot be cut. Therefore, whenever the same partial struct appears twice, both entries are dropped.

Next, to explain the first test, if two partial structs from the same struct are in E, this violates the condition that each struct is cut precisely once by every other plane. Hence, the list E cannot represent a valid way for P_6 to interact with r and it is not stored in S.

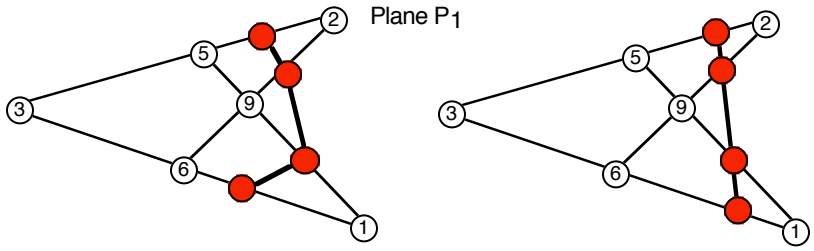

Figure 4.4: A set of joined line segments stretched out to form a straight line

For the second test, a straightforward procedure is executed to ensure that each face cut by the new i-line has exactly two cut partial structs. Interestingly enough, for the graphs in $AG3D_6$, it turns out no element in the set S of lists of partial structs needs to be dropped to meet the requirement. Applying Lemma 5 as explained above also prunes out any list which would violate Lemma 6.

Lemma 6 has two implications. It ensures that the line segment $\overline{ij}$, joining the two new points i and j on a face in w, is not cut by any other i-line in r. Secondly because each cut partial struct is incident on two faces in the planes where it appears, it means the end point of one line segment joins up with the beginning point of the next line segment.

From this it is clear that it is possible to join the pair of new points which must appear on each face in w to form a line segment. Likewise, it is always possible to aggregate the line segments created to form a pseudo-line.

A *pseudo-line* is defined as a simple curve in the plane. In *an arrangement of pseudo-lines*, each pair of pseudo-lines has exactly one point in common and they cross at this common point. The term *stretchable* is used to describe any arrangement of pseudo-lines which can be stretched until they are straight *without changing the adjacency list or the face structure in the plane*. Goodman and Pollack [12] proved that every arrangement of eight or fewer pseudo-lines in the plane is stretchable. Since this study of $AG3D_6$ involves arrangements of only five lines in each plane, it is clear that the new pseudo-line constructed from line segments joining cut partial structs can always be stretched to form a straight line. Figure 4.4 shows an example of the line segments stretched out to form a straight line.

This completes the analysis of correctness for procedures in Step Two and proves that Step Two generates a list of all the valid cuts a new plane P_6 might make through r.

4.2.4 Results for Step One and Step Two

When Steps One and Two are implemented to study the graphs of $AG3D_6$, a set of 59 lists of cut partial structs are generated. However, due to the symmetry of r with respect to $i\text{-}line_2$ and p_{10}, several of these lists are equivalent to each other. In Figure 4.2, the equivalent pairs of points are (p_1, p_3), (p_2, p_4), (p_8, p_9) and (p_5, p_7). Thus for example, if list A consists only of p_2 and p_5 and list B consists of p_4 and p_7, A is equivalent to list B. Note, however, that if B is made up of p_1 and p_7, it cannot be equivalent to A, because p_1 is a degree-3-quad point and point p_2 is a degree-3-tri point. In addition, the discussion in Section 4.3.7 below will show that two more entries are required to represent the cases where the new plane P_6 completely misses $AG3D_5$.

The lists of partial structs establish that a new plane P_6 can cut 3, 5, 6, 7, 8 or 9 partial structs in r. It is impossible for 4 or 10 structs to be cut since there are no lists recorded with these numbers of elements. A list of three partial structs occurs whenever P_6 isolates a degree-3 point from the rest of r. A list of five partial structs occurs when two points, a degree-3 point and a degree-4 point, are isolated.

Once the symmetry of r has been taken into account, the number of entries in S drops to 31. With the two cases to deal with P_6 completely missing r, the final number of lists in S is 33. The following sections will show that each list in S can give rise to more than one distinct graph.

4.2.5 Overview of Step Three - Determining all the equivalence classes of possible planes for r

The goal in Step Three is to determine all the equivalence classes of possible planes that can be constructed from each list s_i in S.

Input: the set S of lists of cut partial structs, and the data structures X, Y and Z (again described briefly in Section 4.3.1 and in detail in Section 4.3.8).

Procedure: Since the procedures in Step Three and the rationale behind them are quite involved, the detailed discussion is postponed until Section 4.3.

Output: 245 graphs of $AG3D_6$.

4.2.6 Step Four - Identifying isomorphic graphs of $AG3D_6$ and the cardinality of $AG3D_6$

Input: the 245 graphs from Step Three.

Procedure: Each graph from Step Three is converted to the specialized adjacency-list format used by Nauty [18], an automorphism software program with procedures to identify equivalence classes of graphs. The output lists the graphs within each equivalence class.

Output: the set of 43 distinct graphs in $AG3D_6$.

4.2.7 Step Five - Checking that each of the 43 graphs is an arrangement graph

Although the set of 245 graphs has been reduced to 43, it is necessary to verify, for each remaining graph, that the 10 new points added to r can all be embedded in a sixth flat plane. If the plane is not flat, then the sixth plane is not supported and graph generated from r and the 10 new points cannot be an arrangement graph.

Input: the set of 43 distinct graphs in $AG3D_6$.

Procedure: A software tool, *ArrangePak-3D* [5], which generates and manipulates arrangements of planes in three dimensions was used for this. In each case, the sixth plane was successfully found. Thus, it is possible to conclude that the cardinality of $AG3D_6$ is 43.

Output: wire models for each distinct graph in $AG3D_6$.
The method used to construct wire models of each of the graphs is as follows:

- Co-ordinates for three of the ten new points are calculated in 3-space that are consistent with the adjacency list for that graph.

- The equation of the possible sixth plane is calculated using these three positions.
- Intersections for each of the seven remaining i-lines with the sixth plane are calculated using ArrangePak-3D and compared with the adjacency list information for that abstract graph.

Drawings of each of these 43 graphs with the vertices on the circumference of a circle are reproduced in Appendix B. The graphs and a wire model with co-ordinates for each point are also listed on a web-site (`www.cs.uleth.ca/~nickle/arrangements`) which can be downloaded in the .gml format suitable for further computational processing.

4.3 The Heart of the Methodology - Details for Step Three

By the end of Step Two, all the different lists of how the partial structs in r could be cut by a potential sixth plane P_6 have been identified and stored in a set of lists S. As mentioned above, the goal for Step Three is to place all the new points resulting from cut partial structs, and then identify all the different possibilities for locating any remaining points (which must be placed in the extremities of the i-lines of r). The potential new plane P_6 must:

- be in general position with respect to the other planes in r.
- cut only the partial structs in r specified by s_i.

Since the set S of lists of cut partial structs is the output of Step Two, the second condition is easy to implement. For each list s_i of partial structs in S in turn, a new point must be assigned to each partial struct in s_i. Then, depending on the orientation of the sixth plane P_6, there is very often more than one possible way for the remaining new points to be placed beyond one or other of the extreme ends of the uncut structs in r. This can be seen in Figures 4.5 and 4.6 where the same three partial structs are cut but it is clear that there is more than one choice for locating the remaining points in the extremities of the i-lines without partial structs cut. Hence, each s_i may give rise to more than one equivalence class of planes and, in turn, to multiple graphs in $AG3D_6$.

In this study, the approach adopted for the task of identifying each distinct equivalence class of planes is to break this large task into a set of smaller, and somewhat easier, sub tasks. The natural

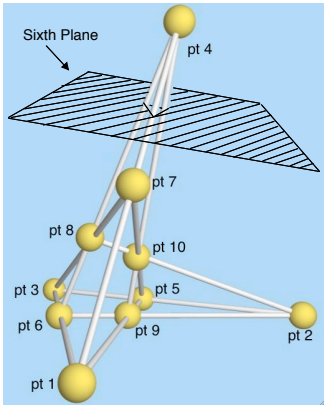

Figure 4.5: Sixth plane P_6 in one position

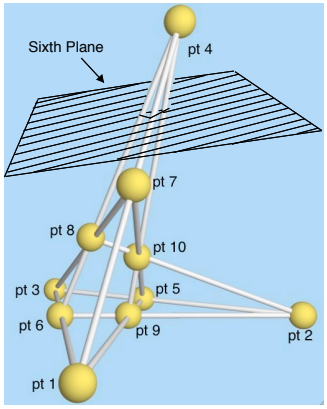

Figure 4.6: Sixth plane P_6 in second position

decomposition to examine, and the method adopted in this study, is the equivalence classes of lines in each plane. This amounts to examining the different ways that an extra i-line can appear in each plane in r. Then, because each i-line is shared between two planes, it is necessary to ensure that the locations of the new points in each plane of r are compatible across *all* the planes of r. Therefore the focus of analysis shifts from the set of planes taken together to examining each plane separately and then ensuring that the changes made to all the planes are compatible.

Section 4.3.2 shows that it is relatively simple to determine how points created from cut partial structs can be joined to form a new i-line in each of the planes of r. It is more challenging to determine how many different ways the rest of the required points can be placed in the extremities of pre-existing i-lines that do not have cut partial structs. Placing the remaining points is dealt with in Sections 4.3.3 through 4.3.7, and determining the number of different ways the points can be placed is discussed in Sections 4.3.6 and 4.3.7. One of the key requirements is to check that the set of new i-lines, placed one per plane, is consistent. This is explained in Section 4.3.5. However, before turning to these matters, the immediately following section, *Preliminaries for Section Three*, provides additional necessary information.

4.3.1 Preliminaries for Step Three

In this section, two definitions are presented to distinguish between points created by cut partial structs and those that are not. Then the way the new i-line is represented is explained, the concept of a variation is discussed, and the data structures used in Step Three are briefly outlined. Full details of the data structures are presented in Section 4.3.8 when the reasons for various features in the structures will be clear.

Different procedures are used to process points which are placed on existing partial structs of r and those that are placed in the extremities of the i-lines of r. Therefore it is useful to differentiate between the two types of points.

Definition: A **pre-fixed point** is one that is located between on a specific cut partial struct $\overline{ij}$ in r, and therefore in the two planes which share $\overline{ij}$.

Definition: A **pre-located point** is placed in one of the extremities of an i-line l and is then restricted to this one extremity. The pre-located point results when an i-line l in one plane of r is assigned a new point and subsequently, when l is considered in the other plane which shares l, the new point for l must remain in the same location. Note that a new point placed in an i-line in a plane of r where the other plane sharing that i-line has not already been analysed is not a pre-located point.

Now turning to how a new i-line is represented, once the points in a plane of r have been joined to form an i-line, the new i-line is represented as a sequence *Seq* of four ordered intersection points — the first point interpreted as the extreme point at one end of the struct on l, followed by the second and third points and the last point as the extreme point at the other end of the struct on l. An intersection point is not identified with co-ordinates; instead, in addition to its position in *Seq*, it is identified by the other pre-existing i-line with which it intersects in the wire model w and where it is located along the pre-existing i-line. Note that the struct on l can also be represented by *Seq*. This is because each point in *Seq* is also a point on the corresponding struct on l. Finally, it does not matter which extreme point is chosen first because a reversed sequence of four points is considered to be the same i-line and struct as the not-reversed sequence. In the discussions below, it will be clear from context whether *Seq* is being used to represent the i-line l or the struct on l. Therefore

Seq is defined as follows:

Definition: The **ordered sequence** *Seq* of four new points represents both the i-line l and the struct on l. Each point represents the intersection of the new i-line l with one of the pre-existing i-lines of the wire model w in r.

In addition, it is necessary to consider the basic conditions that any set of four points must satisfy in order to represent the image of a new plane in a plane of r. First, there must be a new point for each i-line in the plane of r. This is required since all the planes, including the new plane, must be in general position in order satisfy the requirements to form an arrangement. Second, a point placed in a particular segment (either a partial struct or an extremity) of an i-line in one plane must be in the same position in the other plane which shares that i-line. This follows from the fact that each i-line is created by the intersection of exactly two planes. Hence, the *same* i-line appears in exactly *two* planes. Third, it must be possible to join the points to form a straight line. This is necessary from the fact that the image of the new plane in each existing plane is a straight line. The new i-line is added so that it satisfies the conditions for forming an arrangement of five pseudo-lines in that it forms a connected curve and intersects every other pseudoline exactly once. As mentioned previously, it is then guaranteed to be stretchable for sets of less than eight pseudo-lines [12]. This motivates the following definition:

Definition: A **variation** for a potential plane P_i in r is a sequence *Seq* of four points which are placed so that:

- there is a point in *Seq* associated with each of the original i-lines in P_i.
- a point placed in a particular segment of an i-line (either a partial struct or an extremity) must occupy the same position in the other plane which shares that i-line.
- the new i-line constructed from *Seq* and added to the existing i-lines in a plane of r satisfies the conditions for forming an arrangement of pseudo-lines.

The set V of possible variations for one plane in r is clearly the set of *Seq* which represent a new i-line and which are constructed to meet the conditions listed in the definition of variation.

Finally, as mentioned above in the Overview of Step Three (Section 4.2.5), the other inputs to Step Three are data structures X, Y and Z. A brief description of each one is presented below to serve as background material for the following sections explaining how to construct the new i-lines from new points. Once this has been outlined, full details of the data structures are contained in Section 4.3.8.

- Data structure X stores r as a list of structs with each struct, in turn, a list of partial structs. As new partial structs are added to the structs in r, the changes are recorded in X.

- Data structure Y is a set of five lists one for each plane of r. Each list catalogues all the partial structs for a plane of r. For each partial struct $\overline{ij}$, the two faces incident to $\overline{ij}$ are stored as well as four pointers, the previous and next partial structs in each of the faces which are incident to $\overline{ij}$. Data structure Y is used to join new intersection points formed by cut partial structs in the correct order.

- Data structure Z stores information regarding the extreme points on the boundary of the wire model w in each of the planes of r. The four portions of each of the i-lines in each plane can be individually specified in data structure Z so that data structure Z can record where each of the new intersection points is positioned in each plane of r.

4.3.2 Constructing i-lines from intersections due to cut partial structs

This section deals with determining the order of pre-fixed points, or points resulting from cut partial structs, along a new i-line l. For the moment, assume that all the new points on l are the result of cut partial structs. Later, in Section 4.3.4 the new points are not restricted to pre-fixed points.

From Lemma 6, each face f of the wire model w in a plane of r must have exactly two cut partial structs. The two new points, each one placed anywhere along its designated partial struct, are joined. The new line segment which slices across the face in w is a partial struct on the new i-line l for that plane of r.

The full struct on i-line l is constructed using data structure Y. Using the information in Y and starting from one cut partial struct $\overline{ij}$ contained in a face f on the boundary of w, the next partial

struct on the f is assigned to $\overline{ij}$. In this fashion, it is possible to 'walk' around the edge of f to find the other cut partial struct. Once the two cut partial structs on f are identified, a new partial struct can be created by joining the two arbitrary intersection points. After this, the procedure flips to the other face incident to $\overline{ij}$ and continues the same process until the complete new struct is built. Note that the procedure always starts with a cut partial struct on the boundary of w (where the '0' face denotes a partial struct on the boundary of w) so that the struct for the new i-line is built from one side of the wire model w to the other. Also, notice it does not matter which direction is chosen when the process is traversing faces 1, 2 or 3 as shown in Figure 4.7. The procedure will eventually get to the second partial struct either way.

However two complications on the boundary of w can still arise. In Figure 4.7, if the procedure to build l has arrived at p_2 and the direction chosen for traversing the '0' face leads towards p_a instead of p_c, then the next cut partial struct found will be one that is already part of *Seq*. Secondly, if the partial struct associated with p_2 is chosen as the first partial struct on the boundary of w, then p_1 is reached and "walking" along the perimeter of w in either direction does not lead to the correct outcome. Both these impasses are resolved with the help of the following lemmas.

Lemma 7. *If one end of a partial struct s for the new i-line l is identified on the outer boundary of w, the search for the other end of s cannot proceed past a degree-2 point.*

Proof. Without loss of generality, suppose that the process to build the new i-line reaches the partial struct associated with p_2 on the boundary of w in Figure 4.7. Assume that p_2 is identified as one of the ends of s. The challenge is to find where the other end can be located. Assume that the direction chosen to traverse the '0' face in search of the other end of s is towards a, the degree-2 point. The point a, is encountered before the next cut partial struct is found. If the search continues past a, this implies that s must intersect both the i-lines involved in a. This is prohibited because then at least one of two i-lines in w is intersected a second time. The same argument can be used for any degree-2 point in w. Therefore a search *on the outer boundary* for the location of the next cut partial struct in l cannot range farther past a degree-2 point. □

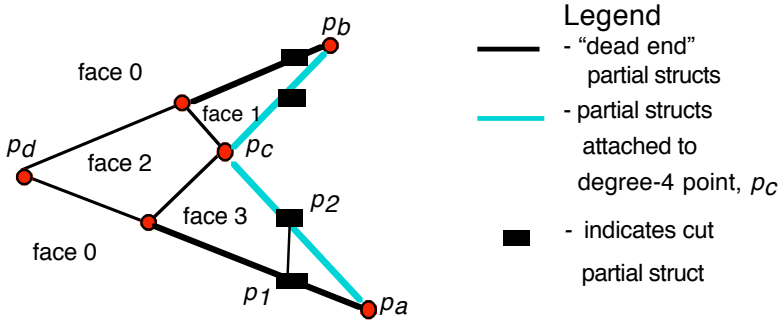

Figure 4.7: Reversing *Seq* if a dead end is encountered

Lemma 8. *If one end p of a partial struct s for the new i-line l is identified on the outer boundary of w, and searches along the boundary of w in both directions from p encounter degree-2 points before another cut partial struct is found, p is the extreme end of the struct on the new i-line l.*

Proof. If the searches in both possible directions from p encounter a degree-2 point before another cut partial struct, then by Lemma 7, it is impossible for the other end of s to be found by proceeding in either of these two possible directions from p. Therefore no other point on the new i-line l beyond p is possible and, from the definition of extreme point, p must be an extreme point. □

Since the next point cannot be found in either direction, the last partial struct accessed must be one of the two dark partial structs shown in Figure 4.7. In this case, a "dead end" has been reached. However, the impasse can be resolved by reversing the elements already in *Seq* and restarting the face traversals from the new end of *Seq*. Then the four elements of *Seq* are found in the correct order.

The pseudocode for the process to join four intersections resulting from cut partial structs in a plane of r is shown below. Recall that a set S of valid lists of cut partial structs is generated at the end of Step Two. The procedure below is used if, there are four cut partial structs contained in s_i which are also contained in the plane of r being analysed. In the procedure, the current struct is always $\overline{ij}$. When $\overline{ij}$ is replaced with the next partial struct in the current face (*currFace*), the direction is from point i to point j.

1: $\overline{ij} \leftarrow$ a cut partial struct which has one of its faces on the outer boundary of w. {This is easily determined because data structure Y records indices for the two faces of each partial struct in w. If one of these faces is the outer boundary, '0' is assigned. Hence, a cut partial struct with '0' as one of its faces is selected.}
2: $reStartBeg \leftarrow \overline{ij}$. {to be used if Seq has to be reversed and the search restarted with the very first entry of $\overline{ij}$.}
3: $Reverse \leftarrow$ false
4: $currFace \leftarrow$ non-zero face of $\overline{ij}$
5: $i \leftarrow 0$
6: **while** $i \neq 4$ **do**
7:
8: **if** $\overline{ij}$ = a cut partial struct **then**
9: Record i-line index of i-line l associated with $\overline{ij}$ and the extreme point on l contained in $\overline{ij}$
10: $i \leftarrow i+1$
11: $currFace \leftarrow$ other face of $\overline{ij}$
12: $reStartCurr \leftarrow \overline{ij}(reversed)$ {Used if the search for the next cut partial struct needs to be restarted in the opposite direction.}
13: $\overline{ij} \leftarrow$ next struct in $currFace$ {The procedure starts "walking" around the face.}
14: **else if** $\overline{ij} \neq$ a cut partial struct AND $j =$ degree 2 AND $currFace = 0$ **then**
15:
16: **if** $Reverse =$ false **then**
17: $Reverse \leftarrow$ true {Indicates this is the first search restart.}
18: $\overline{ij} \leftarrow reStartCurr$
19: **else**
20: Reverse the entries in Seq.
21: $currFace \leftarrow$ zero face of $\overline{ij}$
22: $\overline{ij} \leftarrow reStartBeg$ {Makes the initial entry the new current partial struct.}
23: $Reverse \leftarrow$ false {Resets the flag.}
24: **end if**
25: **else**
26: $\overline{ij} \leftarrow$ next struct in $currFace$.
27: **end if**
28: **end while**

4.3.3 Circular ordering and constructing i-lines from intersections with no pre-fixed points

Recall that w is the wire model of the single graph in $AG2D_5$ in a plane of r. Let l be a new i-line introduced into a plane of r so the intersections that l now makes with the i-lines of w *fall outside* the partial structs of w and so that every i-line in w intersects with l once. Thus, along l, there will be an intersection for each of the four i-lines in w but none are the result of a cut partial struct. The

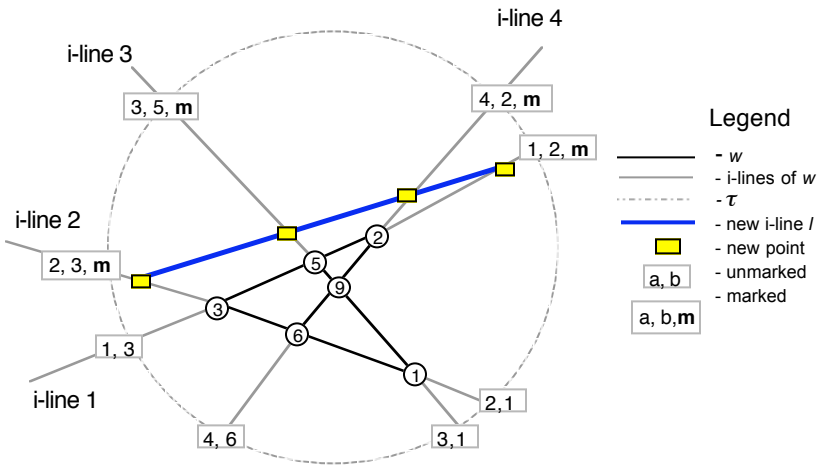

Figure 4.8: The wire model w of plane P_1 and the struct of a new i-line l inside τ with the intersections $\{(2,3), (3,5), (4, 2), (1, 2)\}$

first problem is to identify *the order of these intersections* along the struct of l. The second problem is the fact that there may be more than one way to place the new points along l. Understanding the concept of circular ordering will help build the tools to accomplish both these tasks.

To support the discussion below, let w be contained within a circle τ embedded in the same plane of r, and large enough to contain w and the struct of a new line l inserted in general position. (See Figure 4.8). The set of the i-lines of w is denoted by W. Each i-line in W intersects τ twice, beyond the extreme point at either of its ends. The intersection of each i-line l_a in W with τ is identified by an integer pair, consisting of the i-line index for l_a and the point index of the extreme point on l_a which is adjacent to the intersection on τ.

For each i-line l_a in W which intersects with l outside of its struct, there will be a new point located in exactly one of the two portions of l_a beyond the ends of the struct — in one of the extremities of l_a. Once an i-line in W is assigned a new point, the appropriate intersection on τ is marked to record that this i-line has been processed. Thus each new point on l_a can also be identified by the corresponding integer pair on τ. This is the information required to build the four elements in *Seq*. (See Figure 4.8.)

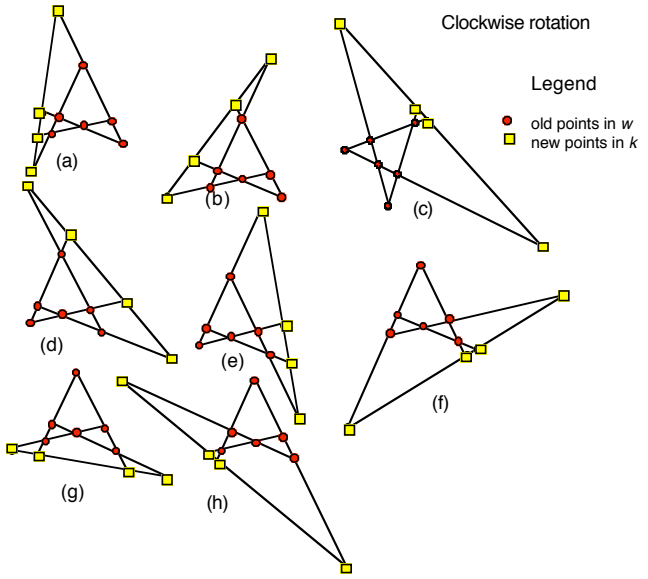

Figure 4.9: The set L of i-lines produced by rotating clockwise around w

In this section, only the two cases where there are no pre-fixed points will be discussed. In Section 4.3.4, the case where there are pre-fixed and pre-located points will be considered.

- **Case 1***: There are no pre-located or pre-fixed new points.*

  Lemma 9. *For each variation, after an arbitrary i-line l_a and an extreme point on l_a are chosen to represent the first intersection of an i-line with τ (and also represent the first entry for Seq), the order of the rest of the entries in Seq duplicates the consecutive clockwise (or counter-clockwise) order of the rest of the i-lines' intersections along the circumference of τ.*

  Proof. (By construction) The struct of the new i-line l must be placed between r and τ. Therefore, the intersections of the i-lines of W with l occur in the same order as the intersections of the same i-lines on τ. It is not possible for the new i-line l to skip one (or more) of the intersections with the i-lines of w, l must also incorporate the intersections in the same order as they are encountered along the circumference of τ. (See Figure 4.8 again.) □

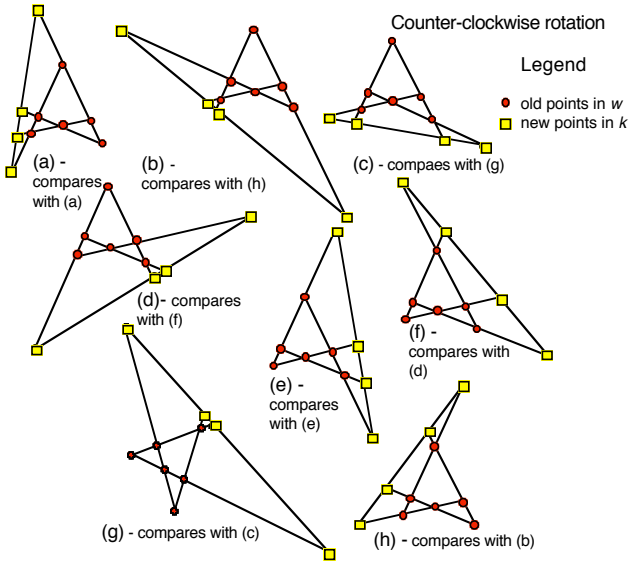

Figure 4.10: The set L of i-lines produced by rotating counter-clockwise around w

Applying Lemma 9. When there are no pre-located new points, then either end of each i-line in W can serve as the initial intersection point. Notice that it does not matter whether we choose a clockwise or counter-clockwise direction - Figures 4.9 and 4.10 show that the set L of possible new i-lines is the same. However, one direction must be chosen to avoid duplication and clockwise is chosen for this study.

- **Case 2**: *There are pre-located points represented by a subset D of the i-lines in W which have new points pre-located in extremities of the i-lines.*

  Incorporating pre-located points. If any of the i-lines of W already have new points located in their extremities, then these points must be incorporated into the new i-line l. To construct l, the intersections on τ for the i-lines in D are marked. Then, a test is run to determine whether these intersections form an uninterrupted block of intersections along the circumference of τ. If not, then the minimum number of intersections are marked to fill the gap. (See Figure 4.11.)

  After that *Seq* is initialized and the points identified so far along the circumference of τ are recorded in *Seq*. It will consist of the integer pairs for each pre-located point and any points

which are forced into a specific location to fill in a gap along τ. If *Seq* has four points, then the i-line is complete. If not, it must be augmented and the points already identified in *Seq* become the core of *Seq*. The rest of *Seq* can be built by adding the intersections with τ for any remaining unmarked i-lines to the appropriate end(s) of *Seq*. The additions are made in the same order as the intersections are encountered along the circumference of τ.

Once again, more than one variation for *Seq* may be possible. Each different *Seq* can be built by either

- using a clockwise rotation from one end point of *Seq* along τ.
- using a counter-clockwise rotation along τ.
- straddling *Seq*.

Figures 4.11 and 4.12 demonstrate several examples. Note that because these are pre-located points, they are always placed in the extremities of the i-lines of W. It is immediately evident that the pre-located points significantly reduce the number of possible variations for *Seq*. These points must always be part of any i-line *l* for that plane of *r*. Procedures for determining the number of variations are discussed in greater detail in Section 4.3.6.

4.3.4 Constructing i-lines from intersections produced by both pre-fixed and pre-located points

The procedures for joining the pre-fixed points have already been discussed in Section 4.3.2. In this case, the same procedures are used but the number of cut partial structs is smaller than four so *Seq* has fewer than four elements. Note that, up to this point, the order of these points in *Seq* is *not* dictated by the order they are encountered on the circumference of τ; instead, it is the order dictated by the procedures for joining points created by cut partial structs.

If *Seq* has fewer than four elements resulting from cut partial structs, then it must have either two or three elements since it is impossible for just one partial struct in any plane of *r* to be cut.

Turning first to the case where two partial structs are cut, this occurs whenever the new i-line cuts the two partial structs associated with any degree-2 point in a plane of *r*. An example is shown

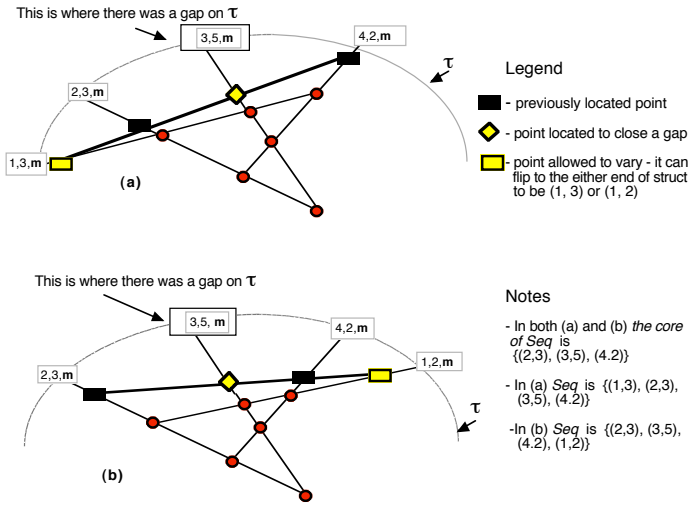

Figure 4.11: An example of the 2 possibilities from two pre-located points with a gap

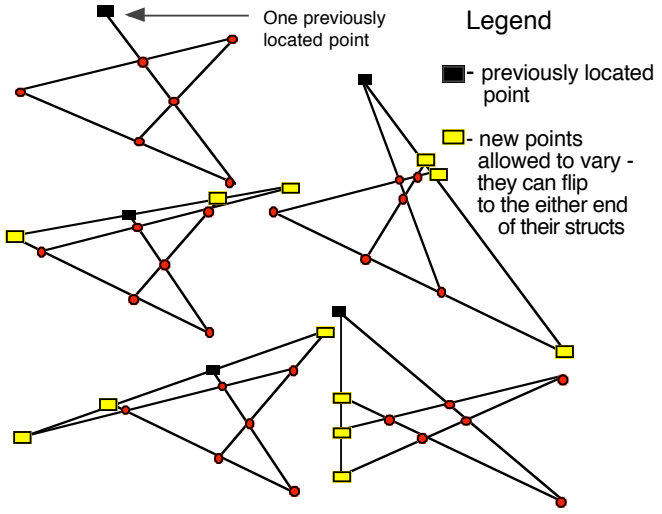

Figure 4.12: An example of the 4 possibilities from one pre-located point

in Figure 4.13. Label these two new points *a* and *b* and imagine that the two points slide in the direction of the degree-2 point until *a* and *b* are re-located in the extremity of their respective i-line beyond the degree-2 point. In this position, they can be considered to be pre-located points. Then the order of the points in *Seq* and the number of different possibilities can be found by applying

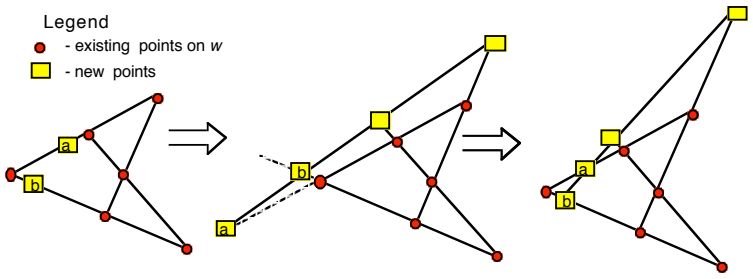

Figure 4.13: Order of points in *Seq* depends on whether points pre-fixed or pre-located

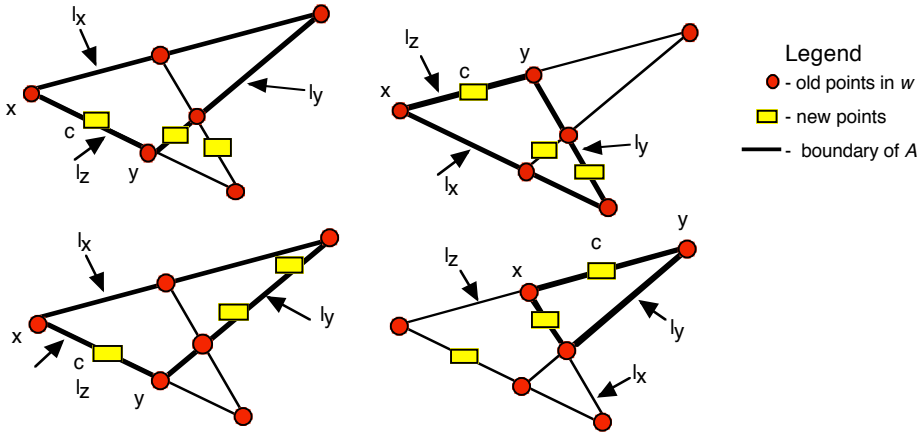

Figure 4.14: Examples where x and y are not degree 4

circular ordering as discussed in Section 4.3.3. To determine the order of the four points *without* a and b re-located, all that is required is to move the new i-line back so that it cuts the two partial structs again. The sole effect is to reverse the order of a and b in *Seq*. (See Figure 4.13.)

Now turning to the case where there are three partial structs cut, there are two sub cases to consider. The first is where the end c of *Seq* represents an intersection with a partial struct s where neither end of s is a degree 4 point. The other is where one of the ends of s is degree 4.

The next lemmas help to answer the question regarding where the next point in l is placed. *Seq* is composed of intersections from three cut partial structs where one end c of *Seq* intersects a partial struct s on i-line l_z. In the first situation, the endpoints of s, x and y, associated with the i-lines l_x

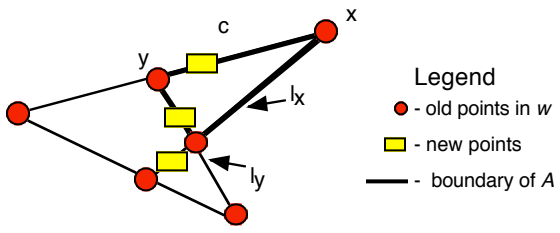

Figure 4.15: Example where the cut partial struct on l_x is contained in w but not in A

and l_y respectively, are not degree 4. Examples are shown in Figures 4.14.

Lemma 10. *When the end c of Seq cuts a partial struct s as described above, exactly one of i-lines l_x or l_y is cut.*

Proof. Because the i-lines are in general position, the three i-lines l_x, l_y and l_z form a triangle A and the sides of A are contained in w. Therefore a new i-line l which intersects with c must intersect either the struct on l_x or on l_y. The i-line l cannot cut partial structs for both l_x and l_y where these partial structs are in A as this would imply a triangle with all three sides cut by a straight line. It is also impossible for l to cut partial structs in l_x and l_y when either the partial struct on l_x or l_y is contained in w but not in A as this would imply that w has a face with only one cut partial struct. See Figure 4.15. □

Alternatively, consider the situation where, without loss of generality, endpoint x is a degree 4 point. Examples are shown in Figure 4.16. Then a new point cannot be placed on the i-line l_x beyond x because this would imply cutting another partial struct in w. Instead, moving in the direction from c to x along the boundary of w, the next partial struct is considered. If its other endpoint (the one that is not x) is degree 2 or 3, then this endpoint x' is substituted for x. Essentially the partial struct s is augmented with a second one so that neither endpoint of s is degree 4.

Lemma 11. *Where the end c of Seq cuts a partial struct s which has a degree-4 endpoint, then s must be augmented as described immediately above. Then exactly one of the i-lines, l_x or l_y is cut.*

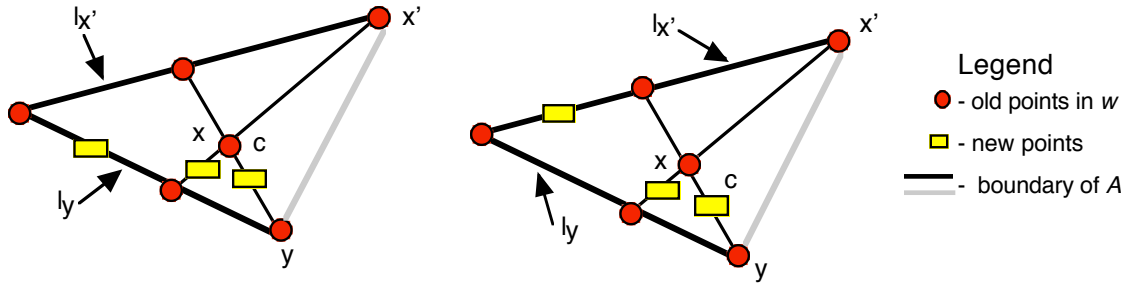

Figure 4.16: Examples where either x or y is degree 4

Proof. The proof is similar to the one in the lemma above. The i-lines l_x, l_y and the line segment joining points x' and y form a triangle A. The line segment $x'y$ is substituted for s and the new i-line l is assumed to cut it once. Once again, an i-line l inserted so that it intersects with $x'y$, must intersect either the struct on l_x or l_y. As above, l cannot intersect with both or it would imply that all three sides of A are cut which is impossible for a straight line. □

Applying Lemmas 10 and 11. The position of the last point for i-line l can be easily found by testing the i-lines of the endpoints of s. Whichever i-line is not already marked on τ represents the next i-line. Now all the required information for adding a new element to the appropriate end of *Seq* is available.

4.3.5 Consistency

Now that the ways to determine the order of points along a new i-line has been discussed, a natural question is whether a valid order always exists. The answer is no. For example, in Figure 4.17, it is impossible for the four pre-located points shown to be joined to form a new i-line l which both respects the constraints imposed by circular ordering and only cuts each i-line once. This leads to the following definition.

Definition: The location of the four new points for the potential plane P_i is *consistent* if a new i-line l can be inserted into P_i while incorporating any pre-fixed and pre-located new points and

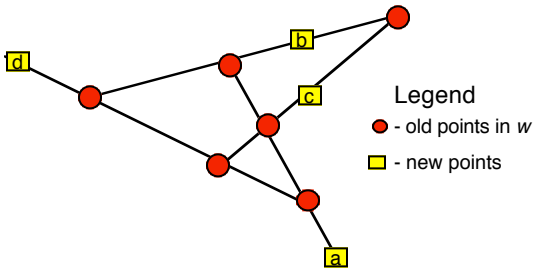

Figure 4.17: The four new points a, b, c and d cannot be joined by a new i-line

respecting the constraints imposed by circular ordering. If it is not possible, the location of the points is *inconsistent*.

If there is an inconsistency in any one of the planes of r then the i-line for that plane cannot be inserted in general position with respect to the rest of the i-lines in that plane. Then the potential new plane P_6 cannot be built and the wire model of that graph in $AG3D_6$ cannot be successfully constructed for that set of new points so it must be abandoned. The points must be placed consistently in *all* the planes for a new arrangement to be formed.

4.3.6 Determining the number of variations possible

Recall from Section 4.3.1 that we define the number of variations in terms of the ways *a new i-line can be added to a plane*. Table 4.1 below summarizes the number of variations which results from different numbers of pre-fixed and pre-located points. The formula for calculating the number of variations when there are no points pre-located or pre-fixed is:

$$2 \cdot (n-1), \text{ where } n = 5 \text{ (the number of planes in } r).$$

The formula for all the other possibilities is:

$$(n-1) - x - y$$

where $n = 5$, x is the number of pre-located or pre-fixed points, and y is the number of points required to close gaps on τ.

Table 4.1: No. of variations for different cases of pre-located and pre-fixed points

Case	Pts. required to close gap on τ	Variations Produced	Examples
A. No points pre-located or pre-fixed	n/a	8	Figure 4.9
B. 1 point pre-located	n/a	4	Figure 4.12
C. 2 points pre-fixed	0	3	Figure 4.18
D1. 2 points pre-located	0	3	Figure 4.19
D2. "	1	2	Figure 4.11
D3. "	2	1	
E. 3 points pre-fixed	n/a	2	Figure 4.21
F1. 3 points pre-located	0	2	
F2. "	1	1	
G1. 1 point pre-located and 2 fixed	0	2	
G2. "	1	1	
H1. 4 points pre-located and/or fixed	n/a	1	

Notes for Table 4.1

A. If there are no pre-located and no pre-fixed points in a plane, there are eight different variations for placing the points (see Figure 4.9 again).

B. For one pre-located point, there are four different ways the i-line can be added. An example is shown in Figure 4.12. Note that there is no corresponding case for one pre-fixed point in a plane of r because this would imply a face in w with only one cut partial struct.

The pre-located point pt can be located at either end of each of the four i-lines in w so pt can occupy any one of eight positions in a plane of r. It must be part of the new i-line l but is not restricted to be in a specific location on l. Hence, it can represent the first, second, third or fourth intersection of the i-lines of w along l.

C. The minimum number of pre-fixed points possible in w is two and occurs when both the i-lines for one of the degree-2 points in w are cut. Since w is symmetric, there are two distinct ways this can occur. Figure 4.18 illustrates the three different positions a pair of pre-fixed points can occupy in the new i-line l depending on which type of degree-2 point is involved.

D1. The i-lines associated with the two new points intersect τ so that there is no gap in the marked intersections along τ. Hence, the two points must appear side-by-side in the new i-line l. In l, they are not restricted to one location. If the four new positions of the points are denoted as a, b, c and d

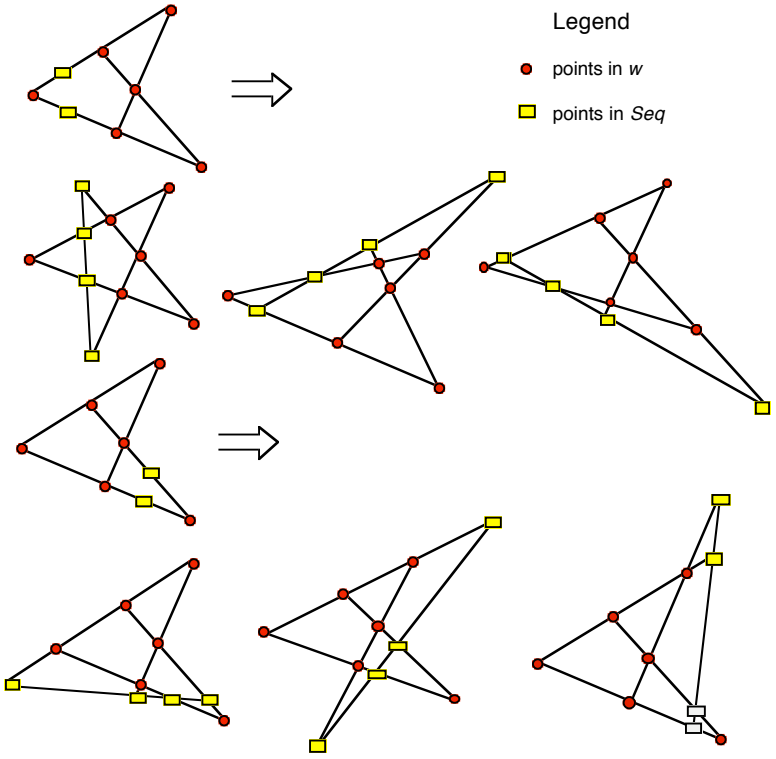

Figure 4.18: Two pre-fixed points produces three variations

(from the extreme point at one end to the other extreme point at the other), the two points can occupy the *ab*, *bc*, or *cd* positions along the i-line. Figure 4.19 illustrates an example. Thus, it is evident that there are three variations possible. Note that this situation differs from the one regarding *two pre-fixed points in w* which is always associated with the i-lines of a degree-2 point in *w*. Here the new points can also be associated with degree-3 points in *w*.

D2. An example of this situation is depicted in Figure 4.11. The analysis is similar to above except now a gap must be closed to make an uninterrupted block of intersections along τ. The number of variations is therefore restricted to two.

D3. In this case, two extra points are required to create an uninterrupted block along τ. Thus the order of all four intersections are specified along l so only a single variation is possible (See Figure 4.20.)

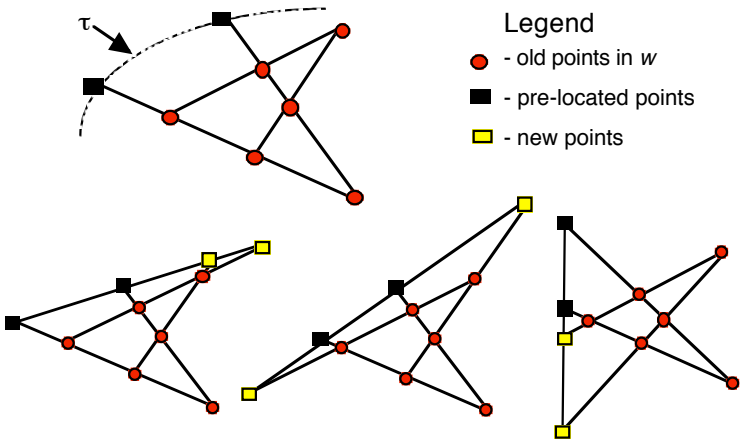

Figure 4.19: Two side-by-side pre-located points in w leads to three variations

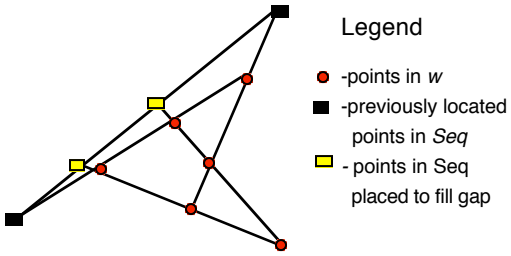

Figure 4.20: If two points are required to fill the gap, only one variation is produced

E. There are three distinct ways that three pre-fixed points can occur in w (See Figure 4.21). In each case, the remaining intersection along l can occur in either one of the extremities of the one uncut i-line in w. Recall that the extremities are defined as the two regions at the ends of an i-line, beyond the extreme points on the i-line. Therefore, for a case with three distinct pre-fixed points, there are clearly two variations. Notice that in these cases, the order of the points created from cut partial structs is not dictated by the order of intersections along τ. Figure 4.22 demonstrates that the gap in τ may not be filled.

F1 and F2. The analysis follows the same approach as in D2 and D3. If the three points are positioned so that either end of the final i-line in w can be used in l there are two variations. Otherwise

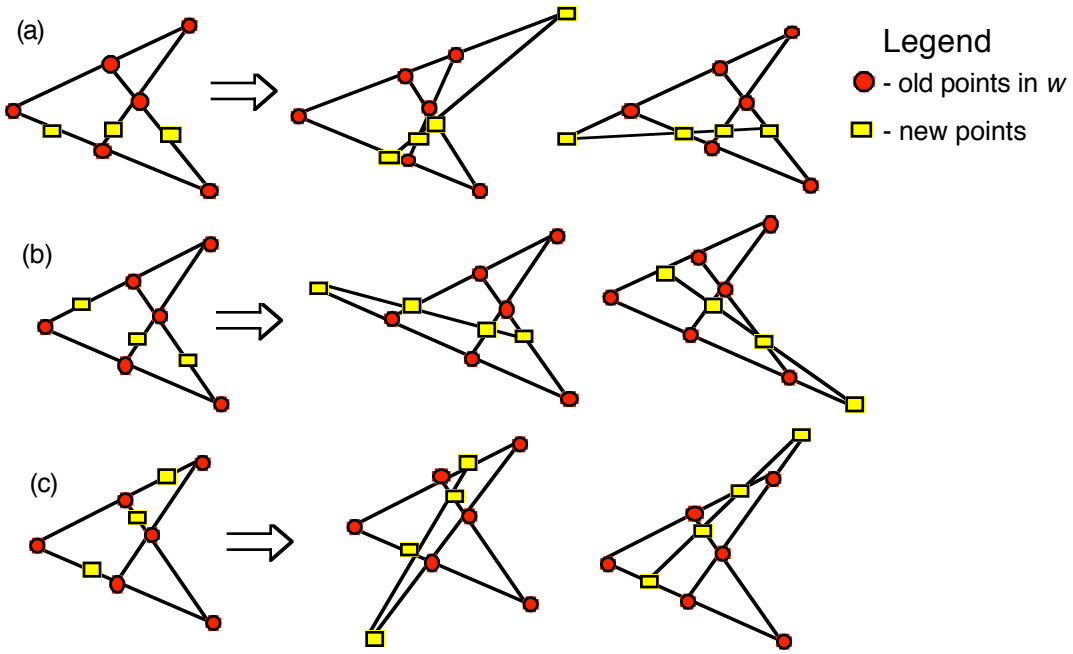

Figure 4.21: Three versions of three pre-fixed points give rise to two variations each

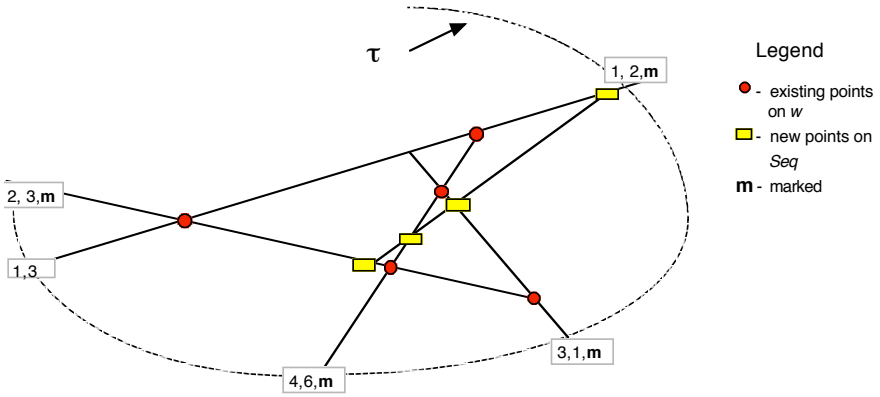

Figure 4.22: If there are three pre-fixed points, the gap on τ is not always filled in

there is only one variation possible.

G1 and G2. Since there is always a minimum of two pre-fixed points in any plane of r, any combination of three pre-located and pre-fixed points means there is one pre-located point and two pre-fixed points. Once again, both the i-lines attached to one of the degree-2 points in w must be involved.

The analysis for this situation is also similar to the one for D2 and D3. Again, if the points are positioned so that either end of the one uncut i-line in w can be used, there are two variations. Otherwise there is only one.

H1. If the four points representing the intersections of the i-lines of w along l are all specified, then the order is also specified and only the single variation is possible.

4.3.7 Number of variations when P_6 misses r

So far, the analysis has been conducted from the perspective that the potential new plane P_6 interacts with r so that some of the partial structs in r are cut. This produces a list of cut partial structs and the set of all these lists is S. However one of the lists that S must include is one which represents P_6 completely missing the wire model r.

This question is approached by considering a case which has already been analysed and is as close as possible to the case where P_6 misses r. The minimum number of partial structs which can be cut in r is three and there are four such cases, one for each of the degree-3 points. Let T be the subset of S where each $t_i \in T$ has exactly three cut partial structs. Each of the equivalence classes of potential planes for each t_i is produced as P_6 "tilts" so that its orientation with respect to r changes and, in turn, the points of intersection of P_6 with the other seven remaining i-lines of r also change. Now imagine sliding an arbitrary P_6 which cuts three structs in r in the direction towards the isolated point until P_6 clears r completely and does not cut any of the partial structs in r. The plane P_6 now intersects the same three i-lines but the new intersection points are placed in the extremities of the i-lines beyond the formerly isolated point. This leads to the following lemma.

Lemma 12. *The number of equivalence classes of planes possible for locating the new points of P_6 when P_6 completely misses r is equal to the sum of the equivalence classes possible for the set of*

four lists, T, where each t_i has exactly three cut partial structs.

Proof. Imagine a potential sixth new plane, P_6, in general position which originally cuts three structs in r. Then imagine moving P_6 in the direction of the one isolated point a, as described above, so that it breaks contact with r. P_6 intersects the three i-lines associated with a beyond a. Now the different ways that P_6 can intersect with the *seven remaining* i-lines are precisely the same ways that P_6 can interact with the *seven remaining* i-lines when the partial structs are cut. This is because the same remaining seven i-lines are all found in the same two planes of r in both cases and these two planes do not contain any of the three i-lines involved in a. Thus, the new i-line l representing P_6 falls outside the wire models embedded in both planes and the number of possible ways that the remaining points can be placed is the same in both cases.

The equivalence classes from the case where P_6 misses r will be different, of course, but only because the intersections for the i-lines which previously had cut partial structs are now located beyond former isolated point, a. Furthermore, from the point of view of equivalence classes of planes, it makes no difference whether P_6 misses a by a wide margin or not. Instead, the focus is on the number of different ways that the remaining seven points can be placed in different portions of the remaining i-lines and this is the same in both cases. This completes the analysis for one of the four degree-3 points in r.

In order to obtain the number of equivalence classes for P_6 missing r, it is therefore necessary to analyse all four degree-3 points of r. However, because r is symmetric with respect to the plane through $i\text{-}line_2$ and pt_{10}, the equivalence classes for P_6 from points p_1 and p_4 will produce sets of graphs which are exactly analogous to each other. Similarly, the sets of graphs for the variations from points p_2 and p_3 will also be analogous. Therefore, for the four degree-3 points, only two new cases need to be considered, one for either p_1 or p_4, and the other for p_2 or p_3. In the algorithm, this is handled by adding two new lists to S, the set of lists of cut partial structs. In one set, the "cut" partial structs are identified as the three partial structs *beyond* p_4, and in the other set, the "cut" partial structs are the ones *beyond* p_3. □

4.3.8 The data structures used in Steps Two and Three

Up to this point, the planes of r have been described using diagrams. Now, the data structures which store the graph and wire model information as combinatorial objects are outlined. With these data structures, it is possible to perform the analysis using computational techniques.

Data structure X

Data structure X is used to store the wire model r and any modifications which are made to r as it is transformed into a graph u in $AG3D_6$.

Data structure X is composed of two two-dimensional arrays. The first two-dimensional array stores the structs of r, where each row represents a struct. The elements in each row are integer pairs and represent end-point indices of the partial structs. The two-dimensional array for r is shown in Table 4.2.

The second two-dimensional array in X stores the four i-lines found in each of the five planes of r. Each row represents a plane and consists of a list of the i-lines embedded in that plane. This second array in X, shown in Table 4.3, is an input to the procedures of Step Three and is not updated.

Table 4.2: First 2D data structure of X for r

Struct	Partial Struct	Partial Struct
1	2 5	5 3
2	1 6	6 3
3	1 9	9 5
4	2 9	9 6
5	3 8	8 7
6	5 10	10 7
7	2 10	10 8
8	9 10	10 4
9	6 8	8 4
10	1 7	7 4

Data structure Y

For each plane of r, data structure Y lists the partial structs contained in the associated 2D wire model w. The relationship of each partial struct to the faces in w is stored and later used to find the order of the points in a new i-line cutting through the existing structs of w. Data structure Y can

Table 4.3: Second 2D data structure of X for r

Plane	Consists of I-lines. . .
1	1 2 3 4
2	1 5 6 7
3	4 7 8 9
4	2 5 9 10
5	3 6 8 10

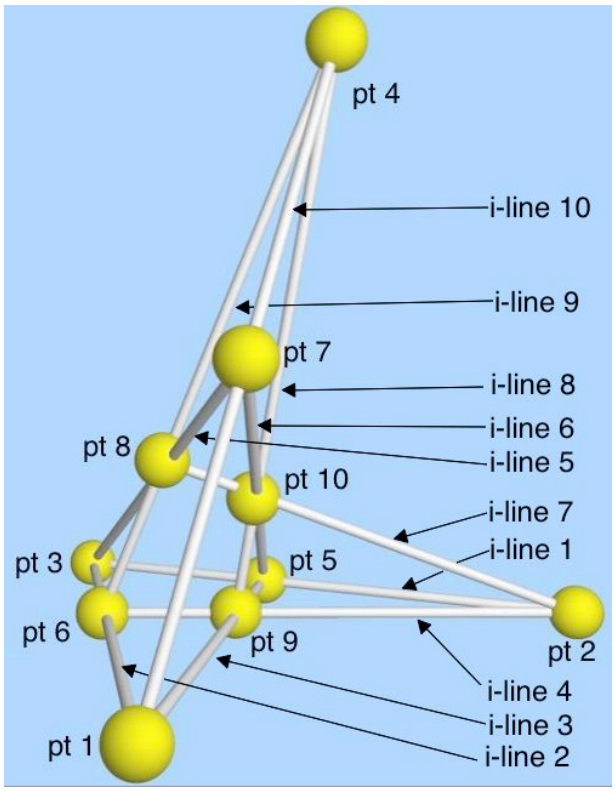

Figure 4.23: The points and i-lines of the wire model r

be completely derived from the initial information in data structure X and is based on Baumgart's *winged-edge* representation [20]. This data structure was chosen because it concentrates on the attributes of the edges (or rather, for this study, the attributes of the partial struct). Each partial

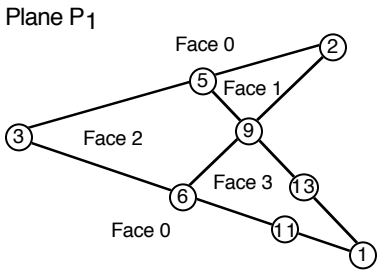

Figure 4.24: Plane P_1 with two new points p_{11} and p_{13}

struct is assigned two faces, and pointers to the previous and next partial structs on each of the two faces. Hence, there are six items associated with each partial struct.

In Sub Step One of the algorithm, when a partial struct $\overline{ij}$ is cut, the existing integer pair is modified by substituting a new index g for one of the endpoints j. Consequently, a new partial struct $\overline{gj}$ is added to Y. The pointers to the next and preceding partial structs for each face are updated. Because data structure Y is only used to connect the points from cut partial structs, it is not necessary for the study of $AG3D_6$ to update it for points located beyond the extreme points of the structs of r. Table 4.4 illustrates the data structure Y for plane P_1 of r with two new partial structs, (11,6) and (13,1), added. The corresponding wire model of P_1 is shown in Figure 4.24.

Table 4.4: Second 2-D data structure of Y for r

Partial Struct	First Face	Next-1st Face	Previous-1st Face	Second Face	Next-2nd Face	Previous-2nd Face
2 5	0	2 9	5 3	1	2 9	5 9
2 9	0	9 13	2 5	1	2 5	5 9
5 9	1	2 9	2 5	2	5 3	9 6
5 3	0	2 5	3 6	2	3 6	5 9
9 13	0	13 1	2 9	3	13 1	9 6
9 6	2	5 9	3 6	3	9 13	11 6
3 6	0	5 3	11 6	2	9 6	5 3
1 11	0	11 6	13 1	3	11 6	13 1
11 6	0	3 6	1 11	3	9 6	1 11
13 1	0	1 11	9 13	3	1 11	9 13

Data structure Z

Data structure Z stores two sets of data. The first set considers the extreme points on the boundary of the wire model w in each plane of r. Each non-degree-4 point on the boundary of w in a plane of r represents an extreme point for at least one struct. A sequence is constructed which consists of the two extreme points of each i-line in the order that they are encountered around the boundary of w. An arbitrary choice is made to have the ends of the sequence represent the degree-2-tri points.

Along with the point index, each element in the sequence also stores the corresponding i-line index and an integer which can take three values: 0, 1 or 2.

- Value "1" indicates the new i-line l intersects with the associated i-line in w *beyond* the extreme point.
- Value "2" indicates l cuts the partial struct adjacent to the point.
- Value "0" denotes that neither occurs - the other end of the struct is the one that interacts with l.

Thus, Z allows the placement of new points in a plane of w to be precisely represented and stored. An example of data structure Z associated with plane P_1 and an arbitrary new i-line l is shown in Figure 4.25. The last sequence of "0's", "1's", and "2's" is referred to as the *location sequence*.

Using the principles of circular ordering and the methods for calculating the number of variations, Z can be used to:

- guide where the new points can be located for the i-lines in w which do not have cut partial structs.
- determine how many different ways the new points for the i-lines in w which do not have cut partial structs can be positioned in w.

The second set of data stored with Z is a set of integer arrays used to establish the validity of Z's location sequences. Recall that some of the sequences produced by placing points are valid

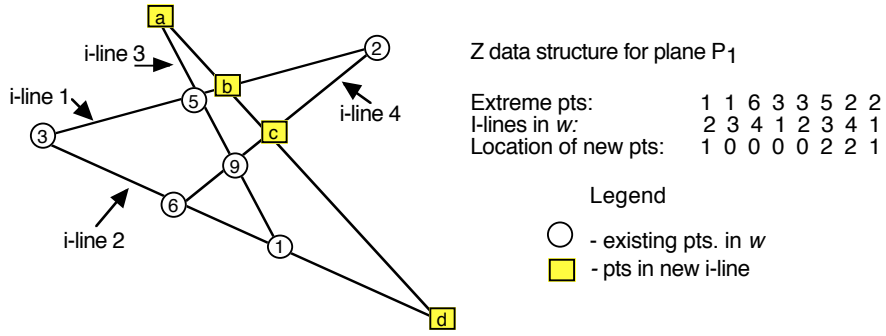

Figure 4.25: An example of data structure Z for P_1 and arbitrary new i-line l

and some are not. Figure 4.26 shows different examples of P_1 with points placed and a number of corresponding Z data structures. Figure 4.26(a) shows one diagram with the points placed in such a fashion that this example would be automatically dropped in Step Two because there is only one cut partial struct on one of the faces. Figure 4.26(b) shows points located in such a way that the principles of circular ordering are not respected. In addition, note that anytime there are two pre-fixed points, two "2's" must appear at either one end of Z or in the exact middle. Then the location of the last two points can be determined from the principles of circular ordering.

Determining the list of valid sequences is not difficult because there is only one distinct wire model w in a plane of r and it is symmetric. Thus the second set of information contained in Z corresponds to all the valid location sequences possible for w in a plane of r. The total list is shown in Table 4.5.

4.3.9 Algorithm for the sub steps in Step Three

Recall that Step 3 runs from within a loop processing each list of cut partial structs s_i of S. Hence, the inputs for Step 3 are the current s_i and the three data structures X, Y and Z. There are three sub steps in Step Three, 3.1, 3.2 and 3.3. Sub steps 3.1 and 3.2 are called once for each s_i while the functions *NumVariations* and *RunVariations* in Sub step 3.3 may be called more than once for each s_i depending on how many planes in r are required to guide where the remaining points for the potential new plane P_6 should be located.

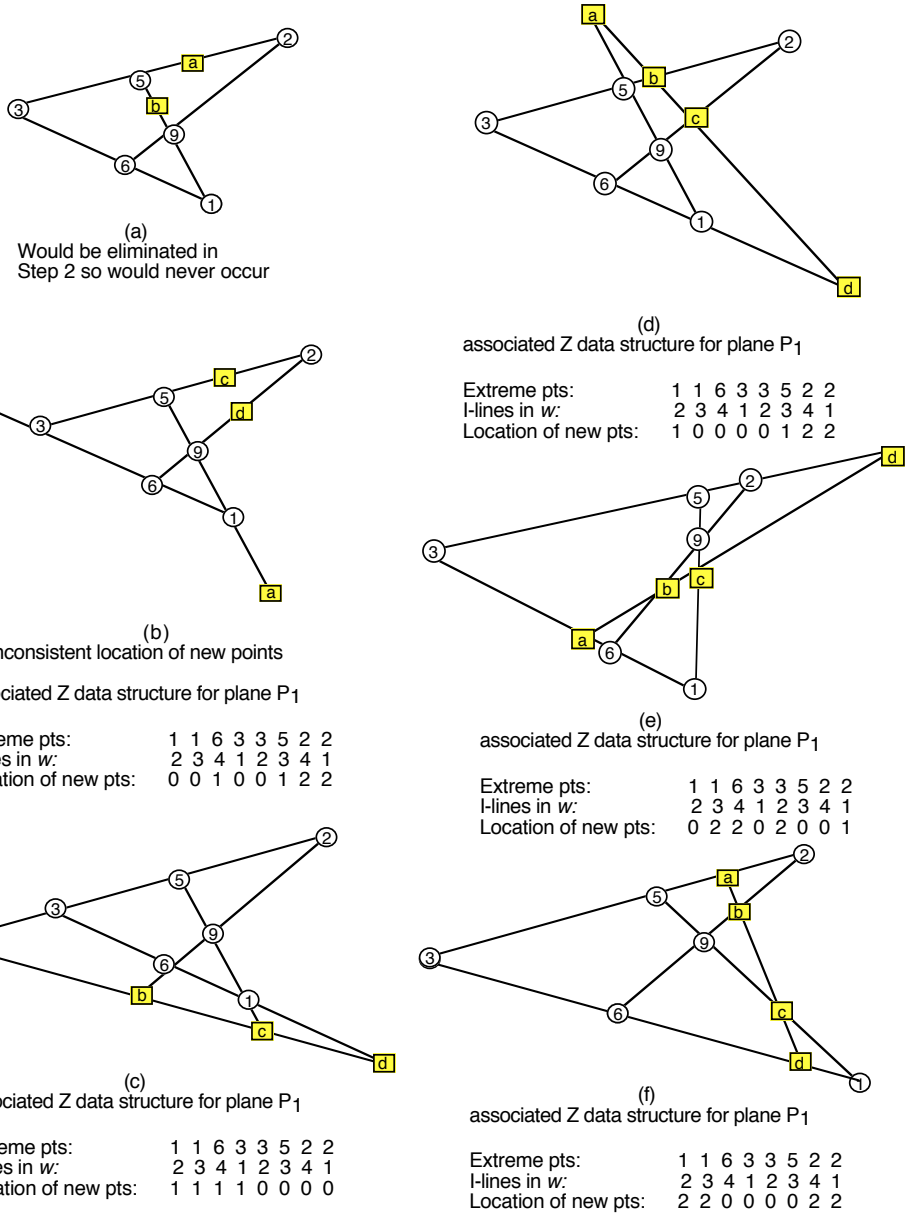

Figure 4.26: For a new i-line l interacting with w in a plane of r, examples of valid and invalid sequences for Z

Sub Step 3.1: - Processing the list of cut partial structs in current s_i

Inputs: Data structures X, Y and Z.

Table 4.5: Valid sequences in the second array for data structure Z

Case	Valid sequences
No partial structs cut	11110000
	01111000
	00111100
	00011110
	00001111
	10000111
	11000011
	11100001
Two partial structs cut	22000011
	22100001
	22110000
	01122000
	00122100
	00221100
	00001122
	10000122
	11000022
Three partial structs cut	02212000
	02202001
	00021220
	10020220
	00002212
	00102202
	21220000
	20220100
	12220000
	02221000
	00012220
	00002221
Four partial structs cut	20200202
	22000022

1: Make copies X', Y' and Z' of the data structures X, Y and Z. {This ensures each s_i can start with a unmodified copy of r.}
2: Data structures X', Y' and Z' are updated with the information for each cut partial struct as listed in s_i.

Outputs: Updated data structures X', Y' and Z'.

Sub Step 3.2: - Selecting a set of planes to guide where the intersections beyond the extreme ends of the i-lines in w are placed

Inputs: Updated data structure X'.

1: Initialize two integer sets $ToUse$ and $NotUsed$.
2: Select the minimum number of planes in r so that each i-line without a cut partial struct is represented at least once.
3: Record these planes in $ToUse$.
4: Record any plane not used in $NotUsed$.

Outputs: Two sets of planes — $ToUse$ and its complement $NotUsed$.

Sub Step 3.3: Constructing all the possible graphs for s_i, a list of cut partial structs

This sub step uses two functions *NumVariations*, which takes parameters $ToUse[Counter], V, X'$ and Z', and *RunVariations*, which takes parameters $ToUse, NotUsed, Counter, V, X', Y'$ and Z'.

NumVariations(ToUse[Counter], V)

1: Select the i-lines in the plane $ToUse[Counter]$ which do not have new points.
2: Identify the different ways the new points can be located and store them in V.

RunVariations(ToUse, NotUsed, Counter, V, X, Y, Z)

1: **for all** v_i in V **do**
2: Make a copy Z'' of data structure Z'. {The reason for this is explained in the *Notes* below}
3: Update data structure Z'' with the new points in v_i
4: **if** $Counter < |ToUse|$ **then**
5: {not Base Case}
6: Initialize a new V'.
7: $Counter \leftarrow Counter + 1$
8: $NumVariations(V', ToUse[Counter])$
9: $RunVariations(ToUse, NotUsed, Counter, V', X', Y', Z'')$
10: **else**
11: {Base Case}
12: **if** the location of the points for all the planes in r is consistent **then**
13: Construct the potential new plane using the positions of all the new points
14: **if** the location of all the points in the potential plane is consistent **then**
15: Make a copy X'' of data structure X'.
16: Update X'' with the new partial structs.
17: Using X'', output the new graph in $AG3D_6$ in a format Nauty can read. {Note that if, for any plane, the location of the points is not consistent, the function does not output the graph for that v_i.}
18: **end if**
19: **end if**
20: **end if**
21: **end for**

Notes: The reason for making the copy Z'' from Z' follows from the fact the algorithm executes a loop for all the possible variations for that plane. Therefore, each time a layer in the recursive

function returns, it does not necessarily return control to the previous recursive call. Instead, if there are variations which have not been implemented, the program "rolls back" the most recent changes in Z' and proceeds with the next variation.

Then Sub Step 3.3 combines these two functions in the following procedure.

Inputs: ToUse, NotUsed, X, Y, Z

1: $Counter \leftarrow 0$
2: Initialize V.
3: $NumVariations(ToUse[Counter], V)$
4: $RunVariations(ToUse, NotUsed, Counter, V, X, Y, Z)$

4.4 Results for $AG3D_6$

4.4.1 Cardinality

Once the 245 wire models are input into Nauty, the Nauty program returns the result that there are 43 distinct graphs. Then, after a wire model of one graph for each of the 43 equivalence classes has been constructed, it can be concluded that the cardinality up to isomorphism of $AG3D_6$ is 43. Instances of two wire models, one an instance of Graph No 12 and the other an instance of Graph No. 38 in Appendix B, are shown in Figures 4.27 and 4.28.

4.4.2 Hamiltonicity

All the 43 graphs in $AG3D_6$ are Hamiltonian. To establish this, each graph was tested using a software package called Groups & Graphs [17]. In Appendix B, the Hamiltonian cycles are shown as the outer boundary of the vertices.

4.4.3 Planarity

All the 43 graphs in $AG3D_6$ are non-planar. Each graph was tested using a software package called OrthoPak [3] which relies on the LEDA [19] library of combinatorial and geometric computing algorithms. The test in this package checks to see if the graph contains either a K_5 subgraph or a $K_{3,3}$ subgraph. If it does, then by the well-known Kuratwoski-Pontryagin theorem, the graph cannot be planar.

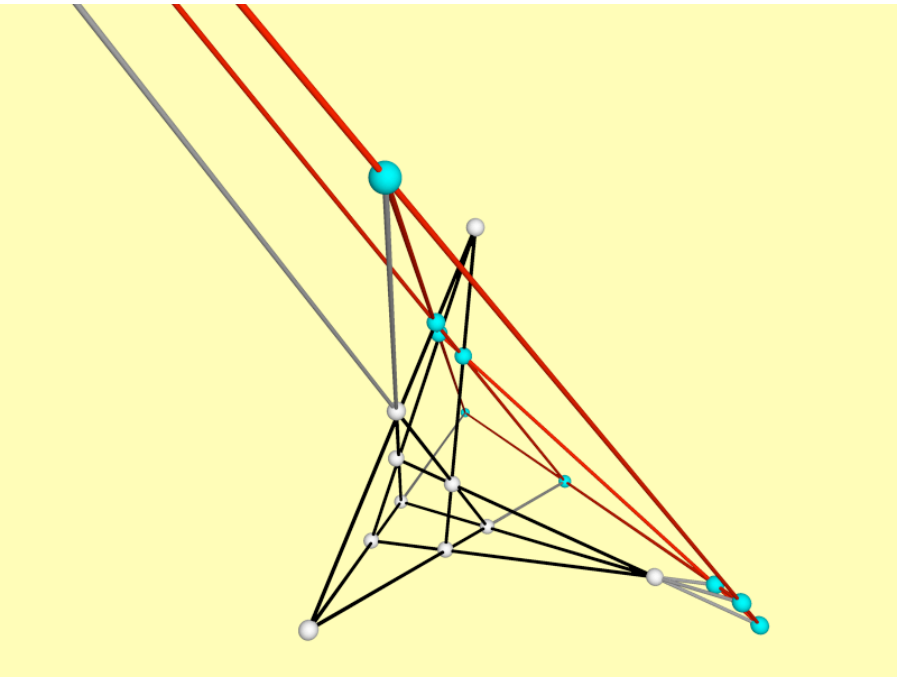

Figure 4.27: An instance of $AG3D_6$ Graph No 12

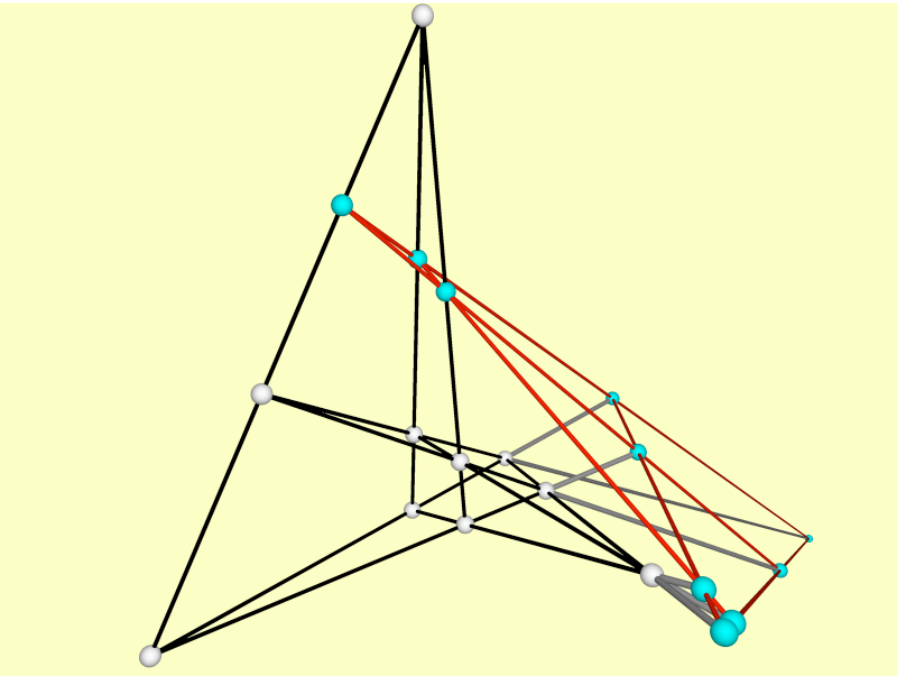

Figure 4.28: An instance of $AG3D_6$ Graph No 38

4.4.4 Summary

The results from the foregoing sections can be summarized by the following theorem.

Theorem 4. *$AG3D_6$ contains 43 distinct graphs. Each one is Hamiltonian and non-planar.*

Chapter 5

Discussion and Open Problems

Basic properties of three different classes of arrangement graphs have been investigated in this thesis. The main results are:

- Cardinality: The cardinality up to isomorphism of $AG3D_4$, $AG3D_5$ and $AG3D_6$ is 1, 1, and 43 respectively. It is clear that establishing the cardinality of $AG3D_6$ is considerably more complicated than for the other two classes. For future research, a natural direction is to investigate the cardinality of $AG3D_7$ and arrangement graph classes with higher numbers of planes. The methodology used to determine $AG3D_6$ and the computational procedures for analysing $AG3D_6$ could be adapted with minimal modifications for analysing $AG3D_7$. This main modification would be to the data structures Y and Z. Data structure Y would need to be adjusted to accommodate the five portions (three partial structs and two extremities) in each i-line of the underlying wire model of $AG3D_6$. The valid sequence information in data structure Z would need to be updated with valid sequences for the wire models of the graphs in $AG2D_6$.

  One significant difference is that the analysis of $AG3D_7$ will be considerably more complex. There are a number of different graphs in $AG3D_6$ and each one can be the underlying foundation for a wire model of a graph in $AG3D_7$. In addition, for $AG3D_7$, there would be 15 new points to place instead of the 10 new points required for $AG3D_6$.

  There is no reason that the methodology for analysing $AG3D_6$ could not be applied beyond $AG3D_7$. Unfortunately, however, it is not clear how the methodology could be employed for

$AG3D_9$. This is because the stretchability of a set of pseudo-lines *in a plane* is not guaranteed when the number of pseudolines is greater than eight [12]. Clearly sets of consecutive line segments could be derived which meet the required criteria for dividing the points in a plane P_i into two subsets. However, there would be no guarantee that the resulting set of line segments could be stretched out to represent the new (straight) i-line resulting from a newly inserted plane in P_i. Further research might uncover a solution to this problem.

- Hamiltonicity: Each of the classes of $AG3D_4$, $AG3D_5$ and $AG3D_6$ is Hamiltonian. It does not seem unreasonable to conjecture that all $AG3D_n$ are Hamiltonian, even though it is known that not all $AG2D_n$ are Hamiltonian. Contrary to 2D arrangement graphs, where there are degree-2 points, the minimum degree for a point in a $AG3D_n$ graph is three. Therefore, there are many more possibilities in 3D arrangement graphs for avoiding the sorts of problems Bose et al.[2] encountered when trying to build Hamiltonian cycles in 2D arrangement graphs.

- Planarity: The classes of $AG3D_4$, $AG3D_5$ are planar but no graph in $AG3D_6$ is planar. There is a graph theory result limiting any planar graph to a maximum of $3x-6$ edges where x is the number of vertices. The number of edges in 3D arrangement graphs, given by the formula $(n \times (n-1) \times (n-3)/2)$ where n is the number of planes, increases more quickly than the number of vertices whose formula is $(n \times (n-1) \times (n-2)/6)$. Therefore it seems reasonable to investigate whether there is a cut-off number c where 3D arrangement graphs with more than c number of planes are non-planar.

  However, note that in any 3D arrangement graph the maximum degree of any vertex is at most 6. Thus, because each edge is counted twice when the sum of the degrees of all the vertices is determined, the number of edges must be $<$ 3x. Furthermore, each 3D arrangement graph must be realizable as a 3D wire model. In a wire model of a 3D arrangement graph, each of the two extreme points of each struct has maximum degree five because, otherwise, it could not be an extreme point. As a result, it is unlikely that 3D arrangement graphs have more than $3x-6$ edges. Consequently, in this thesis, whether 3D arrangement graphs are planar or non-planar as the number of planes increases could not be determined using the $3x-6$

formula. It remains an open question whether $AG3D_n$ with $n > 6$, are planar or non-planar.

In addition, there are a number of directions where further research into arrangement graphs might be pursued which are unrelated to the properties discussed above or the number of planes in the arrangement. One is whether a method could be developed for deciding whether the potential new plane is in fact a plane without having to construct a wire model of each of the non-isomorphic classes of graphs for $AG3D_n$ where $n > 6$. Another is the *3D Arrangement Graph Recognition* problem. It is known that the 2D Arrangement Graph Recognition problem is NP-hard but there is no known analogous result for 3D arrangement graphs.

Bibliography

[1] A. Bjorner, M. Las Vergnas, B. Sturmfels, N. White, and G. Ziegler. *Oriented Matroids*. Cambridge University Press, 1993.

[2] P. Bose, H. Everett, and S. K. Wismath. Properties of arrangement graphs. *International Journal of Computational Geometry and Applications*, 13(6):447–462, December 2003.

[3] M. Closson, H. Everett, S. Gartshore, and S. K. Wismath. Arrangepak, orthopak and vispak 2.0. Technical Report TR-CS-01-98, Computer Science Department, University of Lethbridge, Canada, T1K 3M4, `www.cs.uleth.ca/~wismath/packages/`, 2002.

[4] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry Algorithms and Applications, Second Edition*. Springer Verlag, 2000.

[5] Ray Dufresne and S. K. Wismath. Arrangepak-3d user's manual. Technical Report CS-02-04, Computer Science Department, University of Lethbridge, Canada, T1K 3M4, `www.cs.uleth.ca/~vpak/gd/indexA3D.html`, July 2004.

[6] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1987.

[7] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.*, 15:341–63, 1986.

[8] D. Eu, E. Guevremont, and G. Toussaint. On envelopes of arrangements of lines. *Journal of Algorithms*, 21:111–148, 1996.

[9] Stefan Felsner, Ferran Hurtado, Marc Noy, and Ileana Streinu. Hamiltonicity and colorings of arrangement graphs. *ACM-SIAM Symposium on Discrete Algorithms*, 24:155–164, 2000.

[10] L. Finschi and K. Fukuda. Generation of oriented matroids - a graph theoretical approach. *Discrete and Computational Geometry*, 27:117–136, 2002.

[11] Edgar G. Goodaire and Michael M. Parmenter. *Discrete Mathematics with Graph Theory*. Prentice Hall, New Jersey, 1998.

[12] Jacob E. Goodman and R. Pollack. Proof of Grünbaum's conjecture on the stretchability of certain arrangements of pseudolines. *J. Comb. Theory, Ser. A*, 29(3):385–390, 1980.

[13] B. Grünbaum. *Convex Polytopes*. John Wiley and Sons, London, 1967.

[14] B. Grünbaum. Arrangements of hyperplanes. In *Proceedings of 2nd Louisiana Conference on Combinatorics, Graph Theory and Comput.*, pages 41–106, 1971.

[15] D. Halperin. Arrangements. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 529–562. CRC Press LLC, Boca Raton, FL, 2004.

[16] M. Keil. A simple algorithm for determining the envelope of a set of lines. *Information Processing Letters*, 39:121–124, 1991.

[17] William Kocay. Groups & graphs — a macintosh application for graph theory. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 3:195–206, 1998.

[18] Brendan D. McKay. Nauty user's guide (version 1.5). Technical Report TR-CS-90-02, Computer Science Department, Australian National University, `cs.anu.edu.au/~bdm/nauty/`, 2004.

[19] Kurt Mehlhorn and Stefan Naher. *The LEDA Platform of Combinatorial and Geometric Computing*. University of Cambridge Press, 1999.

[20] Joseph O'Rourke. *Computational Geometry in C, 2nd Edition*. Cambridge University Press, New York, 1998.

[21] H. Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, LIV:150–168, 1932.

Appendix A

List of Specialized Graphical Software Packages Used

This thesis used a number of software resources, apart from the usual typesetting and diagram drawing packages, to generate 2D and 3D diagrams, to produce and manipulate arrangements of planes, and to test for attributes such as isomorphism, planarity and Hamiltonicity.

- **ArrangePak-3D [5]** - ArrangePak-3D is a software program to create and manipulate arrangements of planes in general position and in three dimensions via a graphical user interface. The algorithms for manipulating graphs rely on a proprietary library called LEDA of algorithms and data-types (`www.algorithmic-solutions.info/leda_guide/index.html`). ArrangePak-3D was developed by Ray Dufresne working under the supervision of Stephen Wismath at the University of Lethbridge. More information on ArrangePak-3D can be found at `www.cs.uleth.ca/~vpak/gd/indexA3D.html`.

  In this thesis, ArrangePak-3D was used to create and manipulate the arrangements to produce the GLuskap images of the two wire models of graphs in $AG3D_6$ in Section 4.4.1. ArrangePak-3D also was used to generate wire models (in point co-ordinate form) for one graph from each of the 43 equivalence classes of $AG3D_6$.

- **GLuskap** - GLuskap is a software program for creating, modifying and displaying graph drawings in three dimensions. The colours of the edges and vertices of graphs, the background and the viewing angle can all be user specified. Graphs can be output in .gml and .mg2 (a special GLuskap file format) and graph images can be exported to various formats including .jpg and POV-Ray scene files. The most recent version of GLuskap (2.4) was written by Breanne Dyck, Sebastian Hanlon and Jill Joevenazzo under supervision by Stephen Wismath at the University of Lethbridge in 2004. More information is available at `www.cs.uleth.ca/~vpak/gluskap/`.

  GLuskap was used to create images of wire models of arrangement graphs which were exported as POV-Ray scene files and reproduced as pictures throughout the thesis.

- **Groups & Graphs [17]** - Groups & Graphs is a software application for graphs, digraphs and their automorphism groups. The main developers were Christian Pantel and William Kocay from the Computer Science Department of the University of Manitoba. More information can be found at `bkocay.cs.umanitoba.ca/G&G/Home.html`.

  Groups & Graphs was used to test each of the 43 non-isomorphic graphs of $AG3D_6$ for Hamiltonicity.

- **OrthoPak [3]** - OrthoPak is a collection of routines for displaying graphs orthogonally in three dimensions. This package also draws heavily from the LEDA library of algorithms and data-types (`www.algorithmic-solutions.info/leda_guide/index.html`). It was developed by M. Closson and S. Gartshore under the supervision of Stephen Wismath at the University of Lethbridge. There is a link to the download for ArrangePak, OrthoPak and VisPak 2.0 at `www.cs.uleth.ca/~wismath/`.

  OrthoPak incorporates the LEDA routine to test for planarity. This planarity test was used in Section 4.4.3 to establish that each of the 43 equivalence classes of graphs of $AG3D_6$ is non-planar.

- **Nauty** [18] - Nauty is a set of procedures for determining the automorphism group of a vertex-coloured graph. It is able to produce a canonically-labelled isomorph of the graph, which is used in isomorphism testing. More information, the software and the manual are available at `cs.anu.edu.au/people/bdm/nauty/`. The Nauty developer is Brendan McKay from the Department of Computer Science, Australian National University in Canberra.

  Nauty was used to analyse the set of 245 possible graphs for $AG3D_6$ and discard any graph which was isomorphic to another one in the set.

- **C++ code** - The author developed code to implement the data structures X, Y and Z as well as routines for maintaining and modifying data in the structures, selecting a set of planes to help guide where new points can be located, cataloguing all the different possible locations for points contained in the potential new plane, testing a set of located points for consistency and constructing a potential new plane using the ten new points. The programming language was C++ and the tool suite and integrated development environment used was Xcode 1.5 running on Mac OS X 10.3.

Appendix B

Hamiltonian Cycles in $AG3D_6$ - Part 1

Below a representation of each graph in $AG3D_6$ is produced with the Hamiltonian cycle shown in bold around the outer edge of each graph. The points numbered 1 to 10 represent the original points of intersection from the original five planes; the points numbered 11 to 20 represent the new points of intersection in the new plane P_6.

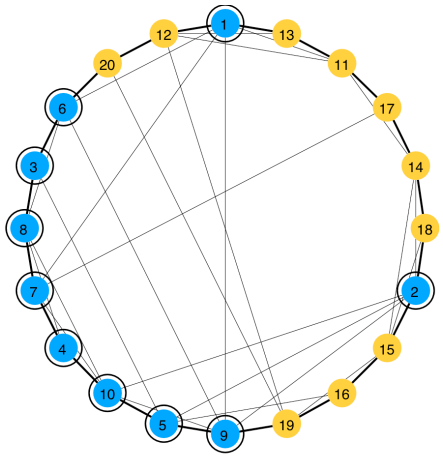

Figure B.1: $AG3D_6$: Graph No. 1

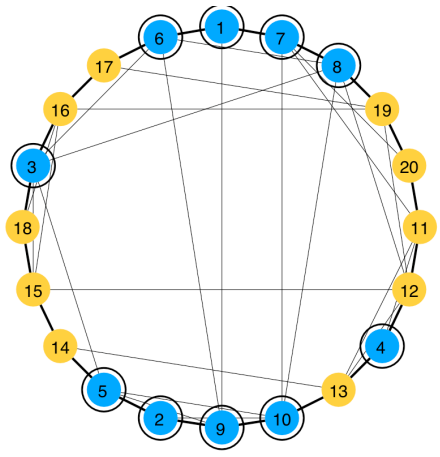

Figure B.2: $AG3D_6$: Graph No. 2

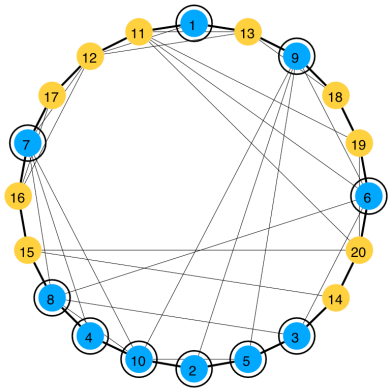

Figure B.3: $AG3D_6$: Graph No. 3

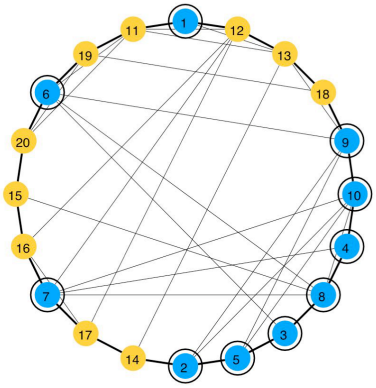

Figure B.4: $AG3D_6$: Graph No. 4

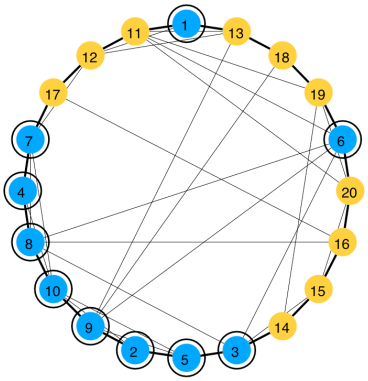

Figure B.5: $AG3D_6$: Graph No. 5

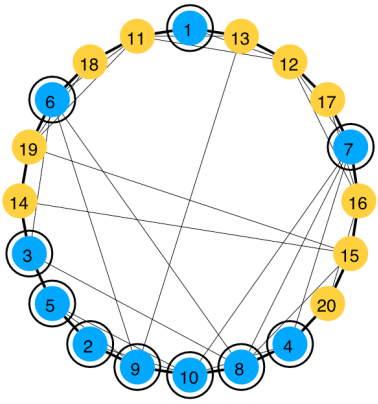

Figure B.6: $AG3D_6$: Graph No. 6

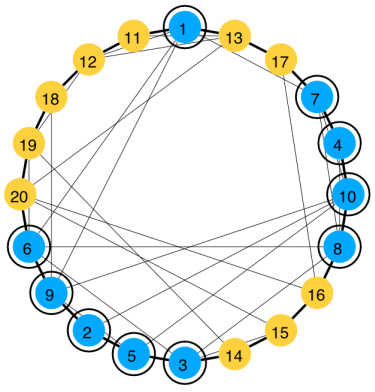

Figure B.7: $AG3D_6$: Graph No. 7

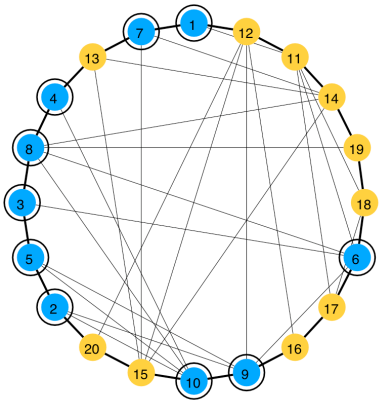

Figure B.8: $AG3D_6$: Graph No. 8

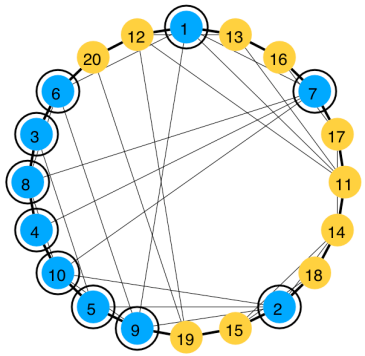

Figure B.9: $AG3D_6$: Graph No. 9

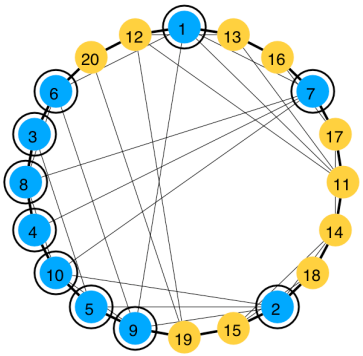

Figure B.10: $AG3D_6$: Graph No. 10

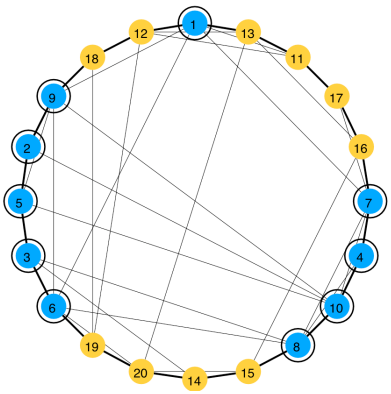

Figure B.11: $AG3D_6$: Graph No. 11

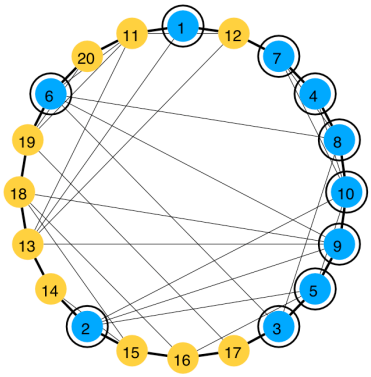

Figure B.12: $AG3D_6$: Graph No. 12

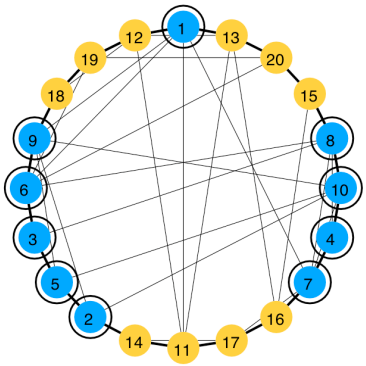

Figure B.13: $AG3D_6$: Graph No. 13

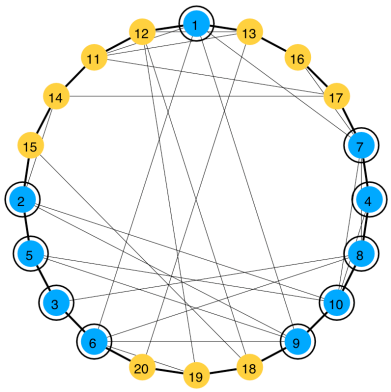

Figure B.14: $AG3D_6$: Graph No. 14

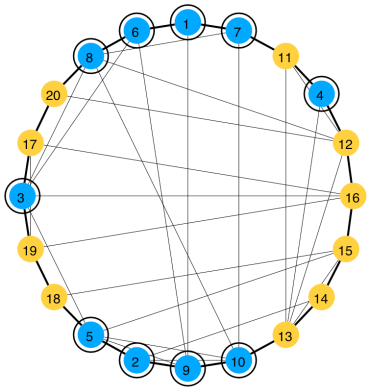

Figure B.15: $AG3D_6$: Graph No. 15

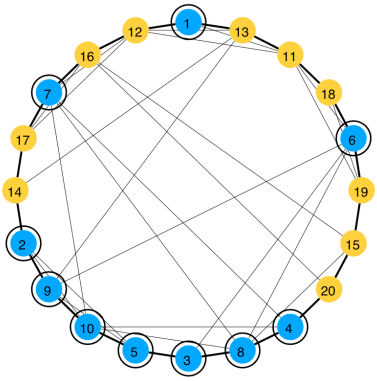

Figure B.16: $AG3D_6$: Graph No. 16

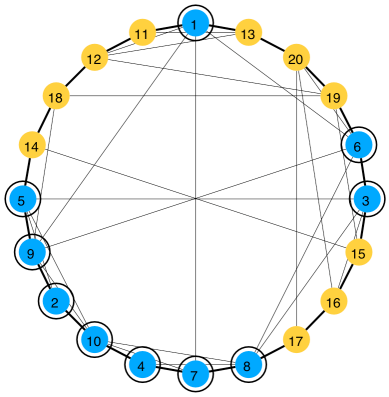

Figure B.17: $AG3D_6$: Graph No. 17

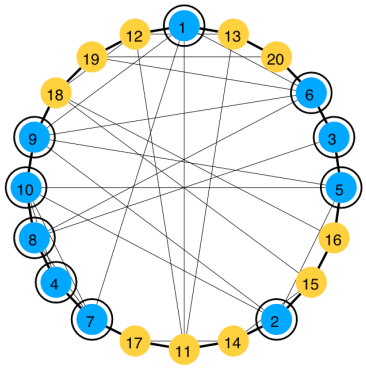

Figure B.18: $AG3D_6$: Graph No. 18

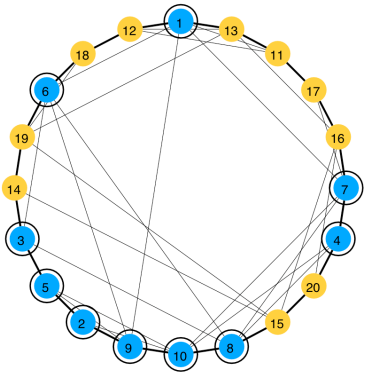

Figure B.19: $AG3D_6$: Graph No. 19

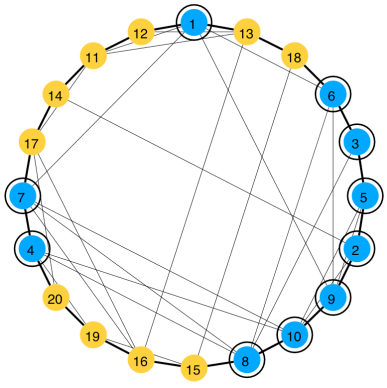

Figure B.20: $AG3D_6$: Graph No. 20

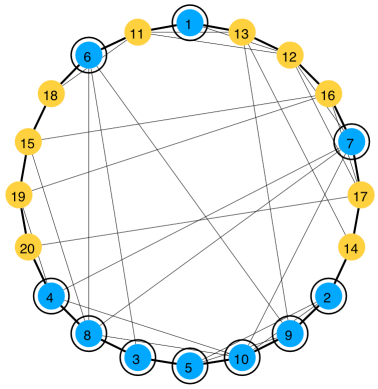

Figure B.21: $AG3D_6$: Graph No. 21

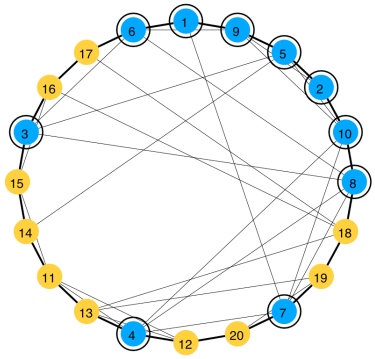

Figure B.22: $AG3D_6$: Graph No. 22

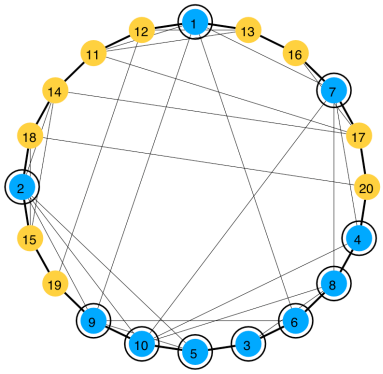

Figure B.23: $AG3D_6$: Graph No. 23

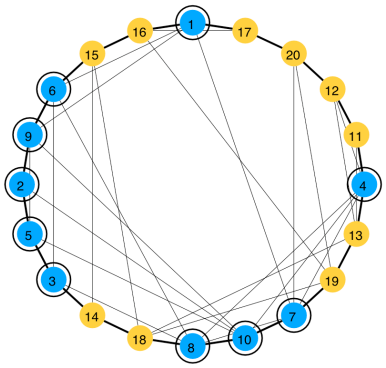

Figure B.24: $AG3D_6$: Graph No. 24

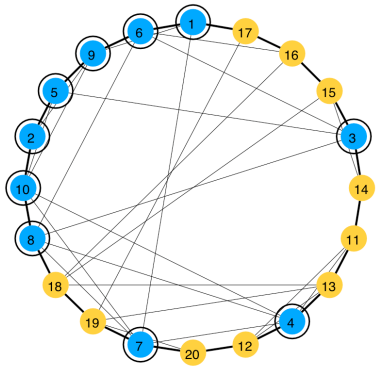

Figure B.25: $AG3D_6$: Graph No. 25

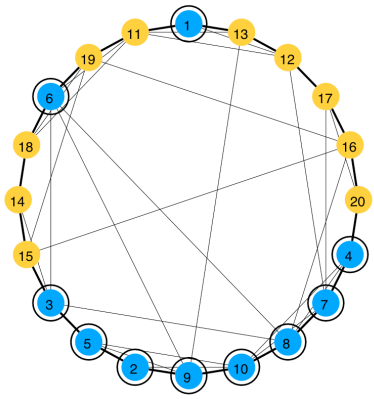

Figure B.26: $AG3D_6$: Graph No. 26

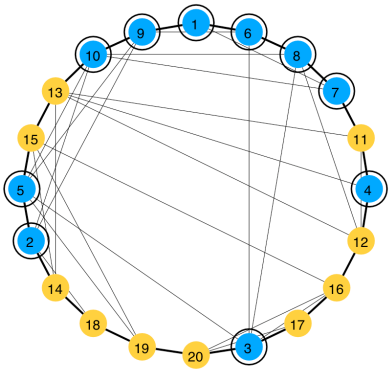

Figure B.27: $AG3D_6$: Graph No. 27

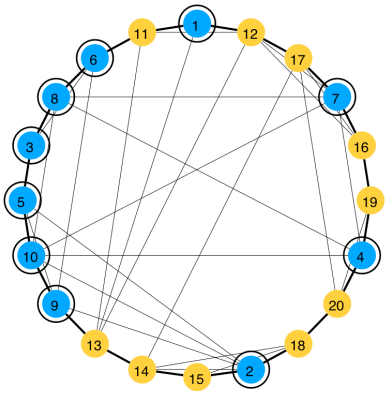

Figure B.28: $AG3D_6$: Graph No. 28

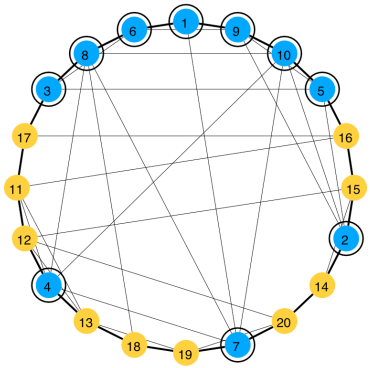

Figure B.29: $AG3D_6$: Graph No. 29

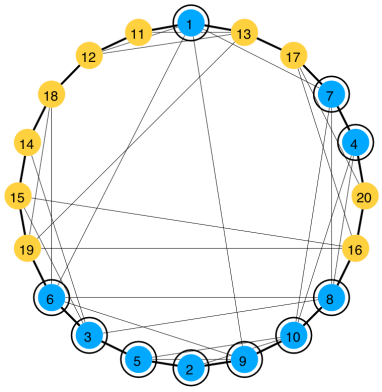

Figure B.30: $AG3D_6$: Graph No. 30

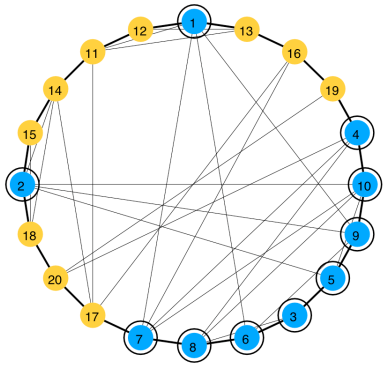

Figure B.31: $AG3D_6$: Graph No. 31

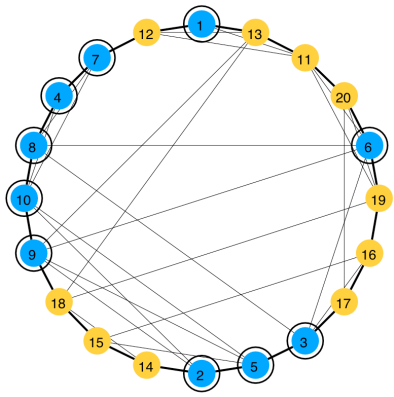

Figure B.32: $AG3D_6$: Graph No. 32

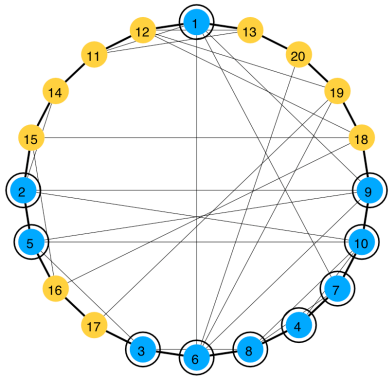

Figure B.33: $AG3D_6$: Graph No. 33

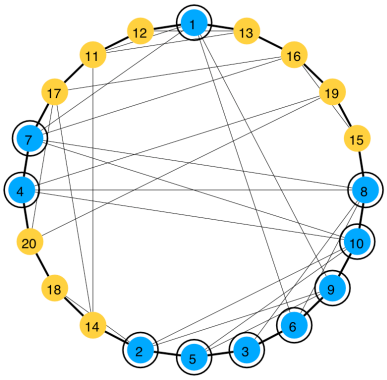

Figure B.34: $AG3D_6$: Graph No. 34

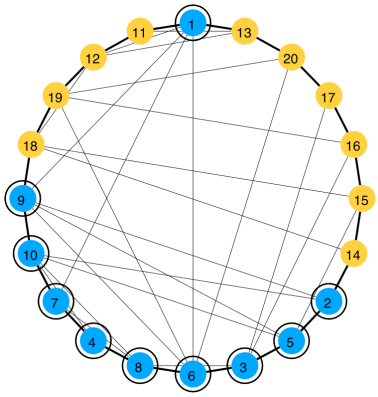

Figure B.35: $AG3D_6$: Graph No. 35

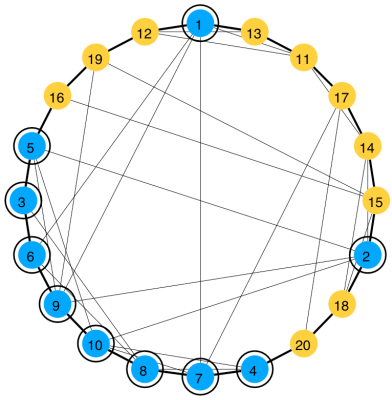

Figure B.36: $AG3D_6$: Graph No. 36

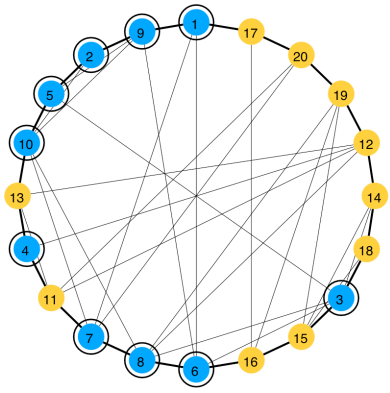

Figure B.37: $AG3D_6$: Graph No. 37

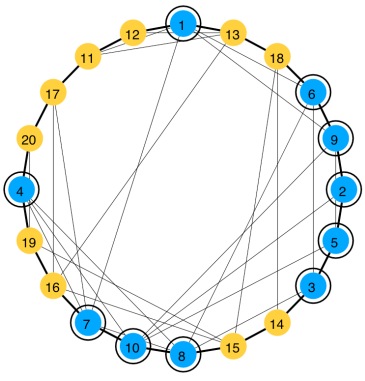

Figure B.38: $AG3D_6$: Graph No. 38

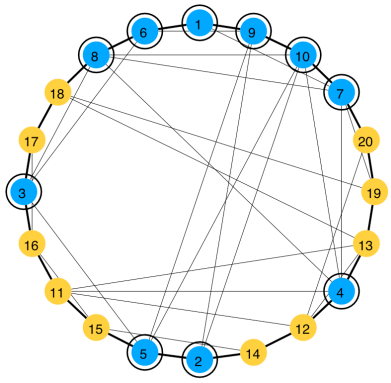

Figure B.39: $AG3D_6$: Graph No. 39

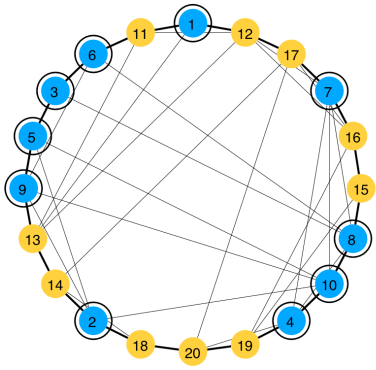

Figure B.40: $AG3D_6$: Graph No. 40

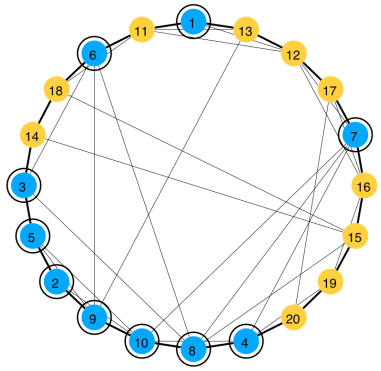

Figure B.41: $AG3D_6$: Graph No. 41

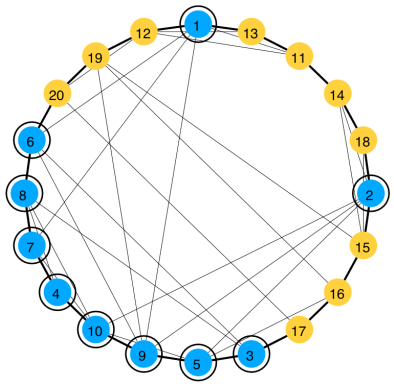

Figure B.42: $AG3D_6$: Graph No. 42

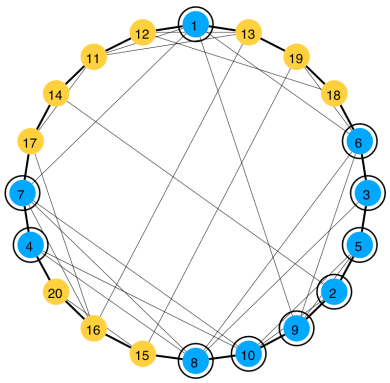

Figure B.43: $AG3D_6$: Graph No. 43