Maximum Clique in Geometric Intersection Graphs

A thesis submitted to the
College of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Jared Espenant

Permission to Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Disclaimer

Reference in this thesis to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement, recommendation, or favoring by the University of Saskatchewan. The views and opinions of the author expressed herein do not state or reflect those of the University of Saskatchewan, and shall not be used for advertising or product endorsement purposes.

Requests for permission to copy or to make other uses of materials in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building, 110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan S7N 5C9 Canada

OR

Dean
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan S7N 5C9 Canada

Abstract

An *intersection graph* is a graph that represents some geometric objects as vertices, and joins edges between the nodes corresponding to the items that intersect. The maximum clique in a geometric intersection graph is the largest mutually intersecting set of objects. In this thesis, the primary focus is to study the maximum clique in various geometric intersection graphs. We develop three results motivated by the maximum clique problem in the intersection graph of disks in the Euclidean plane. First, we improve the time complexity of calculating the maximum clique in unit disk graphs from $O(n^3 \log n)$ to $O(n^{2.5} \log n)$. Second, we introduce a new technique called *pair-oriented labelling*. This method is used to show the NP-hardness of finding a maximum clique in various geometric intersection graphs, acting as a way to augment the commonly used co-2-subdivision approach. Finally, finding maximum clique in two classes of geometric intersection graphs are proven to be NP-hard. These are the intersection graph of disks and axis-aligned rectangles, and the outer triangle graph. The former is previously known to be NP-hard, and so this proof represents the use of pair-oriented labelling in a problem that was otherwise considered difficult to prove NP-hard using a co-2-subdivision approach. The outer triangle graph is a novel intersection graph, which therefore provides new NP-hardness results for finding a maximum clique in geometric intersection graphs.

Acknowledgements

I would like to thank my supervisor Debajyoti Mondal very, very much. He was an excellent support through a process that was more challenging than I expected, and I am glad to have met him. I would also like to thank my wife Aurora who provided the encouragement and patience to continue pursuing my interests. I would like to acknowledge and thank the support from NSERC and the University of Saskatchewan for the very generous CGS-Masters scholarship which has provided me the necessary resources to develop my thesis. Finally, I want to acknowledge J. Mark Keil for a profound amount of guidance and assistance during the journey of my thesis.

I would like to dedicate this thesis to Richard, Almira and Aurora Espenant. Their patience and faith in me to pursue this kind of education is an opportunity I will forever be thankful that they supported me in.

Contents

List of Tables

List of Figures

1 Introduction

The maximum clique problem is one of the most prolific and well-studied problems in computer science ([6], [12], [36], [39], etc). Much like a close-knit group of friends, the clique in graph theory represents the vertices which are all "close" to one another. The maximum clique is a structure observed and studied in *graphs*. A graph consists of a set of points or *vertices* V, and a set of connections or *edges* E which occur between vertices. Often we use the vertices to represent some items, and the edges to represent some type of relationship among these items. Given a graph $G = (V, E)$, a *clique* is a set c of vertices such that any vertex in c has an edge to every other vertex in c. More specifically, one is usually interested in the *maximum clique* of G, which is the clique in G with the most (or tied for the most) vertices.

The notion of finding maximum cliques in graphs has led to applications in multiple topics such as identifying communities in social networks [1], or predicting the loop of a protein structure [38, 8]. The problem however is proven to be NP-complete in general graphs [31]. As a result, researchers are more often interested in specific graph classes for which maximum clique may not be NP-hard, or graph classes in which knowing the time complexity of calculating the maximum clique is of great interest. One such class of graphs is intersection graphs.

An *intersection graph* is a class of graphs that are characterised by defining intersection over some set of elements. Given a set S with n elements, we represent each element with a vertex. For any two elements a and b, if they intersect, then we define the edge $e = (a, b)$. Common examples of intersection graphs include interval graphs, disk graphs, or segment graphs (Figure 1.1).

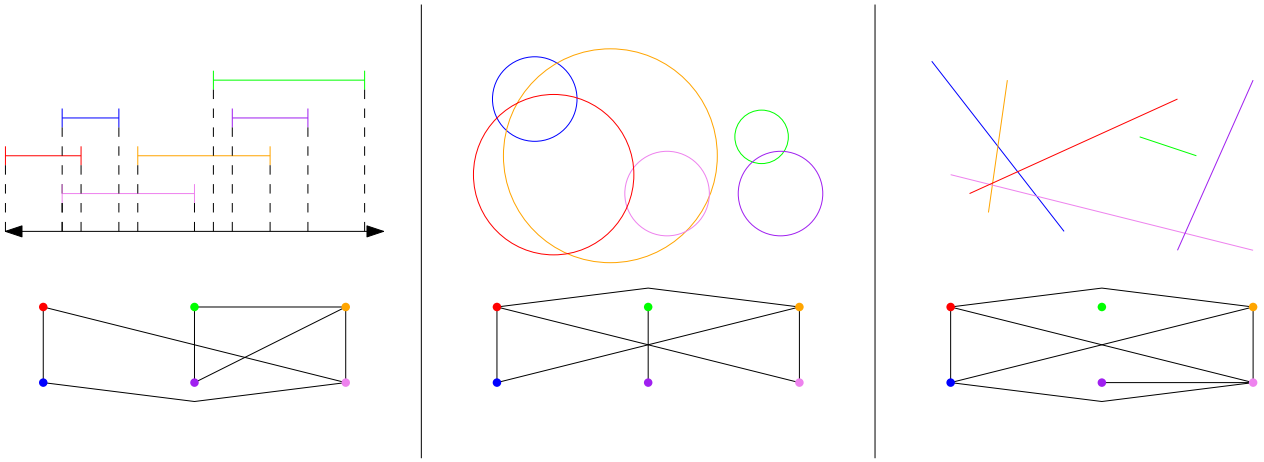

Figure 1.1: Three examples of a geometric graph, and their respective intersection graph. From left to right we have an interval graph, disk graph, and segment graph.

1.1 Motivation

In this paper, the primary focus is on the problem of finding the maximum clique in three different types of intersection graphs. One class of particular interest is the *disk graph*, that is, the intersection graph of disks (filled circles) in the plane. These disk graphs can be used to model specific wireless networks [28], among other applicable problems such as radio networks, sensor networks, and map labelling [21].

The time complexity for finding a max clique of a general disk graph has been an open question for over 20 years [3, 21]. This alone merits a high interest on the topic from a purely curious perspective. Furthermore, this problem remains open even when significant restrictions are placed on the disk graph, such as allowing exactly two types of disks [11], or when the disk radii are restricted to an interval $[1, 1+\epsilon]$ for some $\epsilon > 0$ [4]. It is worth noting however that there exist some PTAS and subexponential-time algorithms for computing the maximum clique in general disk graphs [4, 5]. A PTAS, or *polynomial-time approximation scheme*, represents an algorithm which runs in polynomial time by approximating the solution by a factor of $(1+\epsilon)$, for any given $\epsilon > 0$. Such algorithms may still run slowly, and so variations such as an efficient PTAS (EPTAS), or deterministic PTAS (QPTAS), have also been explored. Bonamy et al. [4] provide a randomized EPTAS, and Bonnet et al. [5] give a $2^{O(\log^5 n)}$-time QPTAS. A subexponential-time algorithm represents an exponential-time algorithm whose exponent grows more slowly. Here, Bonnet et al. [5] develop a $2^{O(n^{\frac{2}{3}})}$-time algorithm. These results represent the continuing fascination and difficulty of correctly solving the long-standing open problem.

Generally speaking, these algorithms represent one of two ways in solving a difficult problem. As seen, one method is to develop algorithms which aim to solve maximum clique in general disk graphs, at the cost of efficient/exact algorithms. The alternative method then is to reduce the complexity of the general disk graph by providing various restrictions, and then developing exact, polynomial-time solutions for these situations. The most common such variation of study is the *unit disk graph*, where all disk radii are of unit length.

For calculating a maximum clique in a unit disk graph, the breakthrough result first came in [14], where Clark et al. showed that a maximum clique can be found in $O(n^{4.5})$-time. The basic idea is to, for each disk pair, use a structure knows as a *lens*, defined as the region of intersection between two disks, to calculate the max clique assuming that the said disk pair is farthest apart in the clique. Later, Eppstein [16] showed how the algorithm could be implemented in $O(n^3 \log n)$-time by leveraging the algorithm found in [2] to maintain a maximum clique throughout the search for each instance of a disk. Beyond this, faster algorithms exist for more constrained setups, such as when the centers of all disks are within a thin horizontal strip [9].

It is worth noting that the use of lenses to calculate the maximum clique does not depend on how the disks are selected, or how the lenses are constructed. For example, the algorithm in [14] takes the simplest approach of selecting all pairs of all disks. It may instead be possible to use the geometry of disk graphs to develop a more efficient way to find or construct the lenses that are used. This line of reasoning acts as the springboard to our first research question:

Research Question 1 (RQ1): Does there exist an algorithm faster than $O(n^3 \log n)$ for calculating the maximum clique for a unit disk graph?

The unit disk graph is one of many intersection graph classes for which polynomial-time algorithms for computing a max clique exist. Some various other results include trapezoid and circle trapezoid graphs [20], intersection graphs of axis-parallel rectangles [29], the complement of outerstring graphs [7], and interval graphs (provided it is an interval representation) [25]. There are similarly various graph classes for which calculating the maximum clique is proven to be NP-hard. Some examples include opposite-angle string graphs [36], ray intersection graphs [12], multiple interval graphs [23], and grounded string graphs [32].

A class of intersection graph are often shown to be NP-hard for computing a maximum clique by utilizing a *co-2k-subdivision* of a graph. Let $\mathcal{I}$ be an intersection graph class. A *2k-subdivision*, where k is a positive integer, of a graph G is obtained by replacing each edge (u, v) of G with a path $(u, d_1, \ldots, d_{2k}, v)$ of $2k$ division vertices. A co-$2k$-subdivision approach takes a graph class for which finding a maximum independent set is NP-hard, and shows that the complement graph of its $2k$-subdivision can be represented by some intersection graph class $\mathcal{I}$ (Figure 1.2).

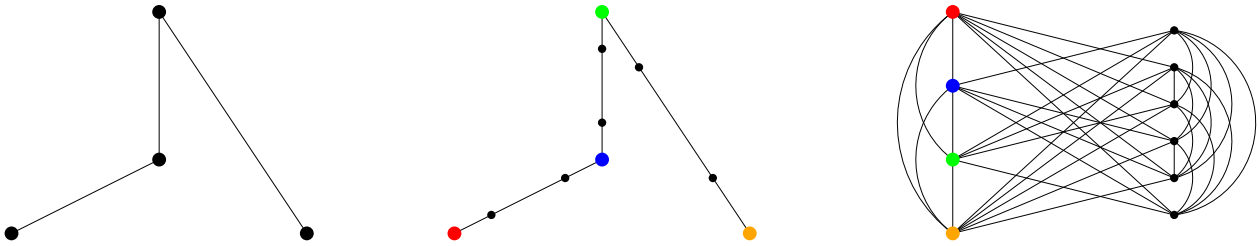

Figure 1.2: An input graph (Left), its 2-subdivision (Middle), and its final co-2-subdivision (Right).

As shown by Chlebík and Chlebíková [13], since the NP-hardness of computing a maximum independent set is preserved by the even subdivision, and since a maximum independent set in a graph corresponds to a maximum clique in the complement graph, this establishes the NP-hardness result for computing a maximum clique in $\mathcal{I}$. Some of the intersection graph classes for which the maximum clique problem has been proven to be NP-hard using the co-$2k$-subdivision approach are: intersection graphs of ellipses and/or triangles [3], filled ellipses and filled triangles [5], opposite-angle string graphs [36], ray intersection graphs [12], multiple interval graphs [23] grounded string graphs [32], and so on.

Despite the widespread usage of a co-$2k$-subdivision approach, there is strong evidence that this method may not be sufficient to prove the NP-hardness of computing maximum clique in a disk graph [5]. This turns our attention to the work of Bonnet et al. [6], in which they attempted to explore intersection graphs of two types of geometric shapes.

Specifically, they examined the problem of calculating the maximum clique of an intersection graph which contains both unit disks and axis-aligned rectangles, or *disk-axis graph*. Bonnet et al. [6] showed that this problem cannot be approximated within a factor of 7633010347/7633010348 in polynomial time, unless P=NP. That is to say, calculating the maximum clique in a disk-axis graph is NP-hard, and even APX-hard. This

result yields two points of interest. The first and most striking of these is that while the unit disk graph and the axis-aligned rectangle graph are individually polynomial-time solvable [14, 29], their combination results in a computationally hard problem. Second, their proof did not use a reduction from co-2k-subdivision, but a new problem called *Max Interval Permutation Avoidance*, or MIPA for short. The basic idea of MIPA is you are given as input a permutation σ on a set of n points represented by a perfect matching between the points $(1,0),(2,0),\ldots,(n,0)$ and $(\sigma(1),1),(\sigma(2),1),\ldots,(\sigma(n),1)$ respectively. You are then given a set of horizontal intervals with integer endpoints. The goal is to place each interval on the line $y=0$ or $y=1$ so as to maximize the total number of edges in the permutation with no endpoint in one of the positioned intervals. We refer the reader to the original paper [6] for additional details.

Clearly, this does not resemble the commonly used co-2k-subdivision approach. This is further compounded by the fact that, as the authors state, the disk-axis graph is "a class for which the co-2-subdivision approach does not seem to work".

The generally prolific and successful nature of co-2k-Subdivision lends itself a desire to standardize various hardness proofs. Therefore it is interesting that there exists a hardness proof which not only uses a reduction from a unique problem, but that the standard way of implementing co-2k-subdivision seems to be insufficient. This produces our second research question:

Research Question 2 (RQ2): Can we leverage the co-2-subdivision technique to prove the NP-hardness of finding the maximum clique for simpler intersection graph classes?

1.2 Contribution

The results of this thesis can be identified by three primary contributions to the computer science community:

Unit Disk Graph (RQ1): We give an algorithm to compute the maximum clique in a unit disk graph (Fig. 1.3 (Left)) in $O(n^{2.5}\log n)$-time, which improves the previously best known running time of $O(n^3\log n)$ [16]. The algorithm is based on a divide-and-conquer approach that, unlike the previous algorithms [14] that search for a clique over all possible lenses, shows how to efficiently merge solutions to the sub-problems to achieve a faster time complexity. Such techniques have previously been used to accelerate computation for other computational geometry problems, for example when finding the closest pair in a point set [37]. This approach however appeared to be highly non-trivial while adapting it for the unit disk graph setting.

Disk-Axis Graph (RQ2): We extend the co-2-subdivision approach to prove NP-hardness for computing a maximum clique in an intersection graph that contains both unit disks and axis-parallel rectangles (Fig. 1.3 (Middle)). This applies the co-2-subdivision approach in a situation which was previously thought to be unlikely [6]. The key idea behind our NP-hardness reduction is to show that every Hamiltonian cubic graph admits a well-behaved edge orientation and edge labeling. That is, its vertices can be labeled and the edges can be oriented such that every vertex has two outgoing or two incoming edges where the labels

of these corresponding neighbors are consecutive. While such orientation and labeling are of independent interest, they allow us to represent the complement of the 2-subdivision of a Hamiltonian cubic graph using a combination of unit disks and axis-parallel rectangles.

Outer Triangle Graph (RQ2): The *outer triangle graph* (Fig. 1.3 (Right)) is the intersection graph of a set of triangles such that all triangles lie between two parallel lines, and exactly one side of each triangle lies on one of the lines. This graph class shares a heavy similarity to the *triangle graph* and *simple-triangle graph*. The triangle graph enforces that one vertex of the triangle lies on one line (the point), and the other two vertices lie on the other line (the interval). The simple-triangle graph adds further restriction, as all intervals must lie on one line, and all vertices must lie on the other line. This lends to their alternate names, PI* and PI graphs respectively, referring to the point-interval structure between the two parallel lines. PI* graphs are known to be NP-complete to recognize [34], whereas PI graphs can be recognized in polynomial time [35]. The outer triangle graph then relaxes the idea that the third vertex of each triangle must line on one of the parallel lines. In this paper, we prove that calculating the maximum clique in this variation is NP-hard by employing the same approach used in the disk-axis graph.

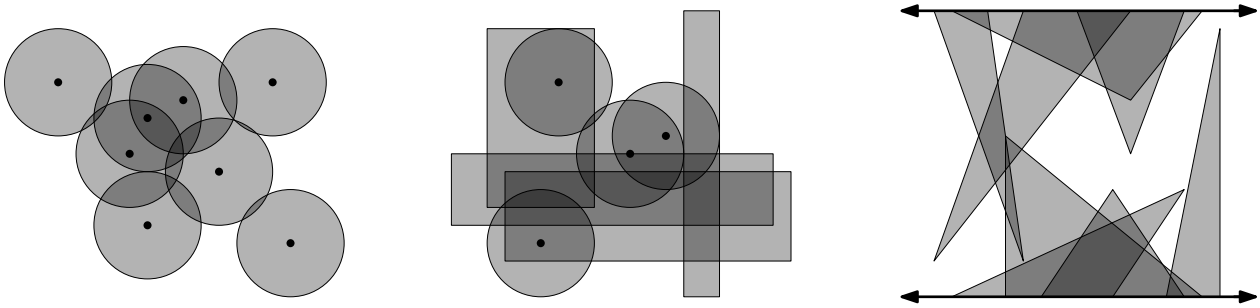

Figure 1.3: An example of each of the three graphs for which results are created. (Left) A unit disk graph. (Middle) A disk-axis graph. (Right) An outer triangle graph.

We present these results through the following sections organized as follows. Section 2 will detail the preliminaries. Section 3 will conduct a literature review on the core ideas for each research question. Sections 4, 5, and 6 each represent the primary contributions to the unit disk graph, disk-axis graph, and outer triangle graph respectively. Lastly, Section 7 will contain the conclusion.

1.3 Declaration

Throughout this document, the term "we" refers to the author and the reader following the theoretical computer science norm for scientific writing. Part of these results have been published in proceedings of The 39th International Symposium on Computational Geometry (SoCG 2023) [17]. The Arxiv revision of the paper can be found here [18].

2 Preliminaries

Here we will introduce some definitions and terms which will be used throughout this thesis. All geometry in this thesis is described in two-dimensional Euclidean space.

Let G be a graph. In this thesis all graphs are assumed to be undirected graphs. The only exception to this is in Section 5 when orientations are applied to edges, forming a directed graph. Unless otherwise stated, we will assume $G = (V, E)$, where V is the vertex set of G, and E is the edge set of G. We will use the notation n and m to denote $|V|$ and $|E|$, respectively. An edge e of G is an unordered pair of vertices in G. That is, if there exists an edge e between vertices a and b, then e may be described as (a, b). We define $\overline{G}$ as the *complement* of G where $\overline{G}$ has the vertex set $\overline{V} = V$. The edge set of $\overline{G}$, is defined as $\overline{E} = \{(a, b)|(a, b) \notin E,\ a, b \in V\}$. Additionally, for two points a and b we denote the *line segment* of a and b as ab. The straight-line distance of ab is denoted as $|ab|$.

In this paper, all graphs we deal with are *simple graphs*. These are graphs which do not contain loops or multiple edges. In another way, the edge (a, a) cannot exist, and the edges (a, b), (b, a) cannot both exist in G. Additionally, Sections 5 and 6 perform constructions in the Cartesian plane, and its associated quadrants 1, 2, 3, and 4 will be referenced accordingly.

The *subdivision* of an edge $e = (a, b)$ is an operation which inserts a new vertex on e between the vertices a and b. If we subdivide e by inserting a new vertex c, the single edge (a, b) will be transformed into the two edges (a, c) and (c, b). A *k-subdivision* of an edge performs the subdivision operation k times on it. We similarly define the k-subdivision of G, meaning each edge of G is subdivided k times. We denote $Sub_k(G)$ as the graph which results from a k-subdivision of G. In a $Sub_k(G)$ graph, the vertices introduced by the subdivision are called *division vertices*, and those which are also in G are called *original vertices*. In this thesis, we are interested in 2-subdivisions. Combining this with the complement, $\overline{Sub_2(G)}$ is the complement of the 2-subdivision of G. This transformation will be relevant in various sections.

We define D_q^r to be the disk D with radius r and center q. For the simplicity of the presentation, the radius will be removed in certain contexts and will instead show D_q to denote a disk with center q. The terms point, circle center, and center will often be used interchangeably depending on the context— particularly in Section 4. Let D_p and D_q be a pair of disks. We define the *lens* of D_p and D_q, written as $L(D_p, D_q)$, to be the region of intersection between the two disks. The *width* of a lens refers to the intersection of line pq and $L(D_p, D_q)$.

The *representation* of a graph G details how the information of the graph is described. For example, we can describe a unit disk graph by its geometric positions- each circle is defined in space with distinct

coordinates. Alternatively, we could describe the same graph G only by its adjacency (say in an adjacency matrix). However, given an adjacency matrix, it may not be obvious if it represents a unit disk graph or not. In this way, the representation of G is important. For the majority of this thesis, we will assume that any graph G is provided in a geometric representation unless otherwise stated. Note that deciding whether a provided graph is a unit disk graph is an NP-hard problem [10], but if the geometric representation of a unit disk graph is given, then its graph representation can be constructed in polynomial-time.

We will now list some relevant graph structures. The *independent set* is a set of vertices such that no vertices in the set are connected by an edge. A *matching* is a set of edges which do not share any vertices. It can be commonly seen as the independent set for edges. A *vertex cover* is a set of vertices such that each edge has an endpoint in this set. The *Hamiltonian cycle*, of a graph is a cycle which visits each vertex exactly once. Similarly, a *Hamiltonian path* is a path which visits each vertex exactly once.

There are some graph classes which are to be defined for the reader. A *bipartite graph* is a graph which can be partitioned into two disjoint sets of vertices such that no two vertices within the same set are adjacent. A *co-bipartite* graph is a graph whose complement is a bipartite graph. A *string graph* is the intersection graph of a set of curves in the plane. A *ray graph* is the intersection graph of a set of rays (half-lines). Similarly there is the *segment graph*, which is the intersection graph of line segments.

3 Literature Review

In this literature review we will take a closer examination on the research questions. There will be two main investigations. First, we will look at previous algorithms which calculate the maximum clique in unit disk graphs. Then, we will look at results which implemented co-2-subdivision as a part of their NP-hardness results for finding maximum clique.

3.1 Unit Disk Graph

Clark et al. [14] gave an $O(n^{4.5})$-time algorithm to compute the maximum clique in a unit disk graph G given a geometric representation. The idea of the algorithm is as follows. Let r be the radius of a unit disk. For each edge (x, y) of the graph, place two disks, $D_x^{|xy|}$ and $D_y^{|xy|}$. That is, $D_x^{|xy|}$ is a disk, not necessarily of unit length, which has the same center as D_x^r. The same is true for $D_x^{|xy|}$. Then, $D_x^{|xy|}$ and $D_y^{|xy|}$ are placed such that their boundaries pass through y and x, respectively. Let S be the set of unit disks with centers in $L(D_x^{|xy|}, D_y^{|xy|})$. Figure 3.1 illustrates an example. Clark et al. showed that the subgraph of G induced by the vertices corresponding to S is a co-bipartite graph $G(S)$. This is because, by the constricting size of the lens, the upper and lower halves of $L(D_x^{|xy|}, D_y^{|xy|})$ each form a clique. Therefore in the complement, the upper and lower halves create two disjoint sets.

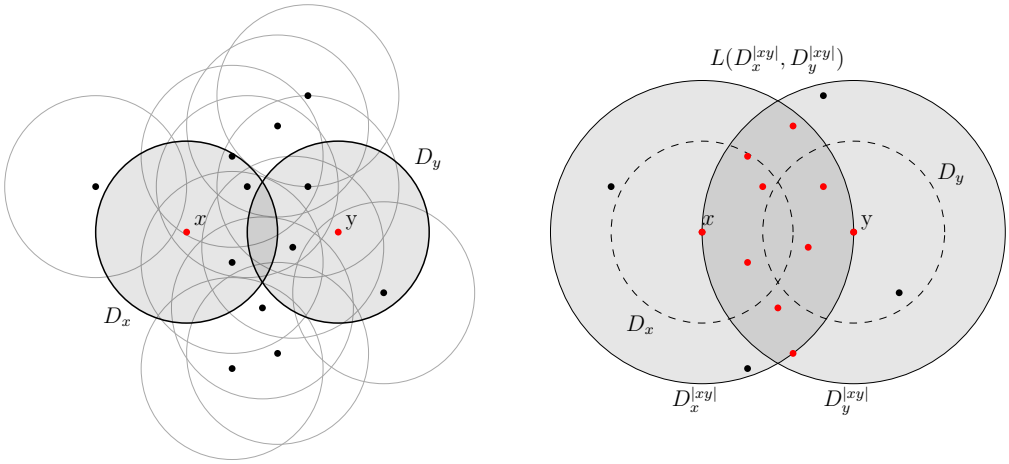

Figure 3.1: (Left) A unit dish graph. Here, disks D_x and D_y are selected to create a lens. (Right) The set S, represented by red dots.

One can thus find a maximum clique in $G(S)$ by computing a maximum independent set in the bipartite graph $\overline{G(S)}$. If (x, y) is a longest edge of a maximum clique M in G, then S must include all the centers of

the disks in M, and $G(S)$ will contain the largest clique in G. This is because any unit disk which is in M but not in S would contradict our assumption that (x, y) have the longest edge in M. Therefore, one can try the above strategy over all edges and find a maximum clique in G. Since G contains at most $O(n^2)$ edges and since a maximum independent set in a bipartite graph can be computed in $O(n^{2.5})$ time by leveraging a maximum matching [27], the running time becomes $O(n^{4.5})$.

Breu [9] observed that Clark et al.'s approach [14] to find a maximum clique can be implemented in $O(n^{3.5} \log n)$ time using a result of Aggarwal et al. [2]. Specifically, Aggarwal et al. [2] showed how to compute a maximum independent set in $\overline{G(S)}$ in $O(n^{1.5} \log n)$ time using a data structure of [26, 30]. Hence, over $O(n^2)$ lenses the running time becomes $O(n^{3.5} \log n)$.

Eppstein [16] observed that while searching through the lenses, instead of computing the maximum independent set from scratch, one can exploit geometric properties to efficiently update and maintain a maximum independent set as follows. For a unit disk center p, let q be a point on the plane such that $|pq| = 2$. Consider two disks D_p and D_q such that their boundaries pass through q and p, respectively. One can now rotate the lens $L(D_p^{|pq|}, D_q^{|pq|})$ around p and update the maximum independent set in the graph corresponding to $L(D_p^{|pq|}, D_q^{|pq|})$ each time a point (center of a unit disk) enters or exists from the lens. An update can be processed by an alternating path search in $O(n \log n)$ time [2]. Since the number of changes to $L(D_p^{|pq|}, D_q^{|pq|})$ is bounded by $O(n)$, the time spent for p is $O(n^2 \log n)$. Hence the overall running time is $O(n^3 \log n)$.

3.2 Co-$2k$-subdivision

The co-2-subdivision approach has been used multiple times when it comes to proving that the maximum clique in a specific intersection graph class is NP-hard [36, 12, 5]. As there are many graph classes for which maximum independent set is NP-hard, there is a high level of selection when it comes to choosing a graph class which can be reduced to the relevant problem for which a researcher wants to prove is NP-hard. Such a reduction usually involves constructing an instance of the intersection graph class in question. We briefly view some intersection graph classes for which co-2-subdivision was used to prove maximum clique is NP-hard.

Middendorf and Pfeiffer [36] were able to prove that finding maximum clique on opposite-angle string graphs, and by extension string graphs, is NP-hard by a reduction on any graph G. The opposite-angle string graph are string graphs in which each string is a line segment with exactly one of two types of bends in it. Since maximum independent set is NP-hard in general [24], this construction follows quickly. The basic idea is to layer strings with the same bend, creating two sets of layered strings, and then intersect the two sets of strings as necessary. Although straightforward, this proof shows the benefit in layering objects so as to permute their accessibility. Similar approaches may be observed in Sections 5 with the disks.

Finding maximum clique in the intersection graph of rays, and by extension segment graphs, was proven to be NP-hard by [12]. The reduction comes from any planar graph G, which themselves are known to

be NP-hard for calculating maximum independent set [24]. The basic construction of the proof involves showing that if the planar graph contains a tree, among some additional conditions, then the graph can be represented as a ray intersection graph by extending the representation of the tree to a new representation. If the planar graph does not have a tree, then the authors prove any planar graph has an even subdivision which decomposes into a tree. Interestingly, the work done by Cabello et al. [12] had settled a 21-year-old open problem posed first by Kratochvíl and Nešetřil [33]. This is a notable result for closing the open problem, but also identifying cases where a more strict graph class is required to use co-2-subdivision.

Bonnet et al. [5] prove that the intersection graph of filled ellipses, and the intersection graph of filled triangles, contains all co-2-subdivisions. That is to say, the co-2-subdivision of any graph can be represented by the intersection graph of filled ellipses, or the intersection graph of filled triangles. This result shows that finding the maximum clique these intersection graphs is NP-hard. The distinction of a filled shape can yield slightly different intersection graphs from that of its non-filled counterpart. The authors highlight this difference by correcting a claim made by Ambühl and Wagner in [3], showing the filled and empty version for ellipses can yield different representations. Still, both the filled and non-filled version of ellipses and triangle intersection graphs all yield NP-hardness.

4 The Disk Graph

In this section we present a $O(n^{2.5} \log n)$ polynomial-time algorithm which calculates the maximum clique of a unit disk graph from its geometric representation. This algorithm uses a divide-and-conquer method, recursively breaking down the problem until reaching a base case. The base case involves generating a valid lens for each candidate disk by leveraging the assurances provided by the recursion. Once the lens is generated, the maximum clique within the lens is calculated as in [14].

4.1 Idea of Algorithm

Here, we present an overview of the algorithm. Let G be a disk graph with n vertices, where each disk is of radius r. Let P be the set of centers of the disks corresponding to the vertices of G. To find a maximum clique we take a divide-and-conquer approach as follows.

We rotate the plane so that no two points are in the same vertical or horizontal line. It is straightforward to perform such a rotation in $O(n^2)$-time [15]. We sort the points in P with respect to their x-coordinates and find a vertical line Z through a median x-coordinate such that at most $\lceil n/2 \rceil$ points of P are on each half-plane of Z. Let P_l and P_r be the points on the closed left half-plane and closed right half-plane of Z, respectively. We will find a maximum clique in P_l and P_r recursively.

Let M be a maximum clique in G. If the set of disk centers corresponding to M is a subset of either P_l or P_r, then such a clique must be returned as a solution to one of these two sub-problems. Otherwise, M contains points in both P_l and P_r. To tackle such a case, it suffices to find a maximum clique in the vertical slab between the vertical lines Z_l and Z_r, where Z_l and Z_r are $2r$ units apart from Z on the left half-plane and right half-plane, respectively. Let $Q \subseteq P$ be the set of points in the vertical slab. Then the maximum clique of G is the maximum clique found over the disks corresponding to the sets P_l, P_r and Q.

Let $T(n)$ be the time to compute a maximum clique in G. Let $F(n)$ be the time to compute a maximum clique in the vertical slab. Then $T(n)$ is defined as follows:

$$T(n) = 2T\left(\frac{n}{2}\right) + F(n). \tag{4.1}$$

We now sort the points of Q with respect to their y-coordinates and find a horizontal line H through the median y-coordinate such that at most $\lceil |Q|/2 \rceil$ points of Q are on each half-plane of H. Let Q_t and Q_b be the points on the closed top half-plane and closed bottom half-plane of H, respectively. We now find a maximum clique in Q_t and Q_b recursively. If the set of disk centers corresponding to M is a subset of either

Q_t or Q_b, then such a clique must be returned as a solution to one of these two sub-problems. Otherwise, each of Q_t and Q_b contains some points of M.

To tackle such a case, it now suffices to find a maximum clique in the square S of side length $4r$ with its center located at the intersection point of Z and H with regions R_1, R_2, R_3, and R_4 (Figure 4.1). Let $B(n)$ be the time to compute the maximum clique in S. Then $F(n)$ is defined as follows:

$$F(n) = 2F\left(\frac{n}{2}\right) + B(n). \tag{4.2}$$

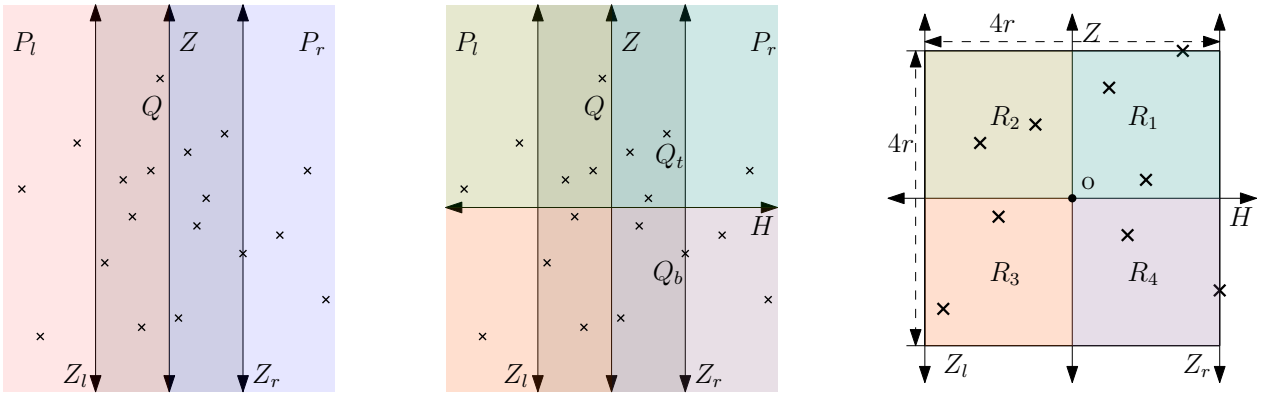

Figure 4.1: Here we detail an instance of the divide and conquer algorithm on P. (Left) The sets P_l (left of Z), P_r (right of Z), and Q. (Middle) Splitting the plane into 4 quadrants with the introduction of Q_t and Q_b, with Q cutting down the middle. (Right) The calculated square S resulting from the previous steps.

In the following, we will show that a maximum clique in S can be computed in $O(n^{2.5} \log n)$ time. Consequently, $B(n) \in O(n^{2.5} \log n)$. By extension, using Equation 4.2 and the master theorem, $F(n) \in O(n^{2.5} \log n)$. Therefore, the time complexity determined by Equation 4.1 is $O(n^{2.5} \log n)$. Note that computing a maximum clique in the square S of side length $4r$ appears to be the bottleneck of our algorithm.

4.2 Square Base Case

The generated square S represents the base case of the algorithm. S is centered at $V \cap H$ and is characterized by four regions: R_1, R_2, R_3, and R_4, each of which are a $2r \times 2r$ square. Observe that the maximum clique of S must include a disk from diagonal regions R_1 and R_3, or from R_2 and R_4. Additionally, let $o = H \cap V$. That is, o represents the centre of S. Please refer to Figure 4.1 (Right) for an illustration. Here we go more in depth as to how the maximum clique M is calculated for a given square S.

The algorithm considers two cases depending on whether all disk centers in S are within a distance of $2r$ from o or not. It processes each case in $O(n^{2.5} \log n)$ time, and then returns the maximum clique found over the whole process.

For the first case, we process disk centers which are within a distance of $2r$ from o. For each such disk center q, the algorithm assumes it is the farthest disk center in M from o, and uses this assumption to

generate a lens which includes M if the algorithm's guess is correct. For the second case, we assume that at least one disk center in M has a distance of more than $2r$ from o. For each such disk center p, the algorithm guesses it is the farthest disk center in M from a unique disk D_a^{2r} with center a. We use D_a^{2r} to order the points by moving its center a along a diagonal path towards o. The stopping position of D_a^{2r} for each p helps generate the lens which includes M if the algorithm's guess is correct. The guesses are verified to be correct by returning the largest max clique found over all guesses.

Each such lens can be generated in $O(1)$ time, shown in more detail in Algorithm ***Square***. Since the corresponding intersection graph is a co-bipartite graph [14], a maximum clique in this graph can be computed in $O(n^{1.5} \log n)$ time [2]. As we are guessing each disk center in S, the overall running time becomes $O(n^{2.5} \log n)$.

This yields the following algorithm, with an associated walk-through in Fig. 4.2:

Algorithm *Square*

1. Assume a square S is given with centre o, a set of disks C, and a maximum clique M with $M \subseteq C$ (Fig. 4.2(a))
2. For each disk centre q which is within or equal to a distance $2r$ from o (Fig. 4.2(b)):
   (a) Define two disks D_q^{2r}, and D_a^{2r} with center a. Then, position D_a^{2r} such that the line segment qa is of length $2r$ and passes through o
   (b) Calculate the maximum clique in the lens $L(D_q^{2r}, D_a^{2r})$ (Fig. 4.2(c))
3. Else there exists one disk center at distance greater than $2r$ from o. Then for each region R_i:
   (a) Define a disk D_a^{2r}. Consider the diagonal d of the region which has o as an endpoint. We place a upon d such that $|ao| = 2r(\sqrt{2} - 1)$. That is, the border of D_a^{2r} touches the end of the other diagonal (Fig. 4.2(d))
   (b) Move D_a^{2r} along the diagonal until a coincides with o.
   (c) For each disk centre p which intersects the border of D_q^{2r} along its sweep in R_i (Fig. 4.2(e)):
      i. Define a disk D_p^{2r}
      ii. Calculate the maximum clique in the lens $L(D_p^{2r}, D_a^{2r})$ (Fig. 4.2(f))
4. Return the maximum clique M found in S

4.3 Proof of Correctness

Here we will prove the validity of Algorithm ***Square***, as well as construct the complete algorithm to find the maximum clique in a unit disk graph in $O(n^{2.5} \log n)$ time.

Let G be a unit disk graph. Consider a disk D_c^r with radius r and centre point c which is contained in

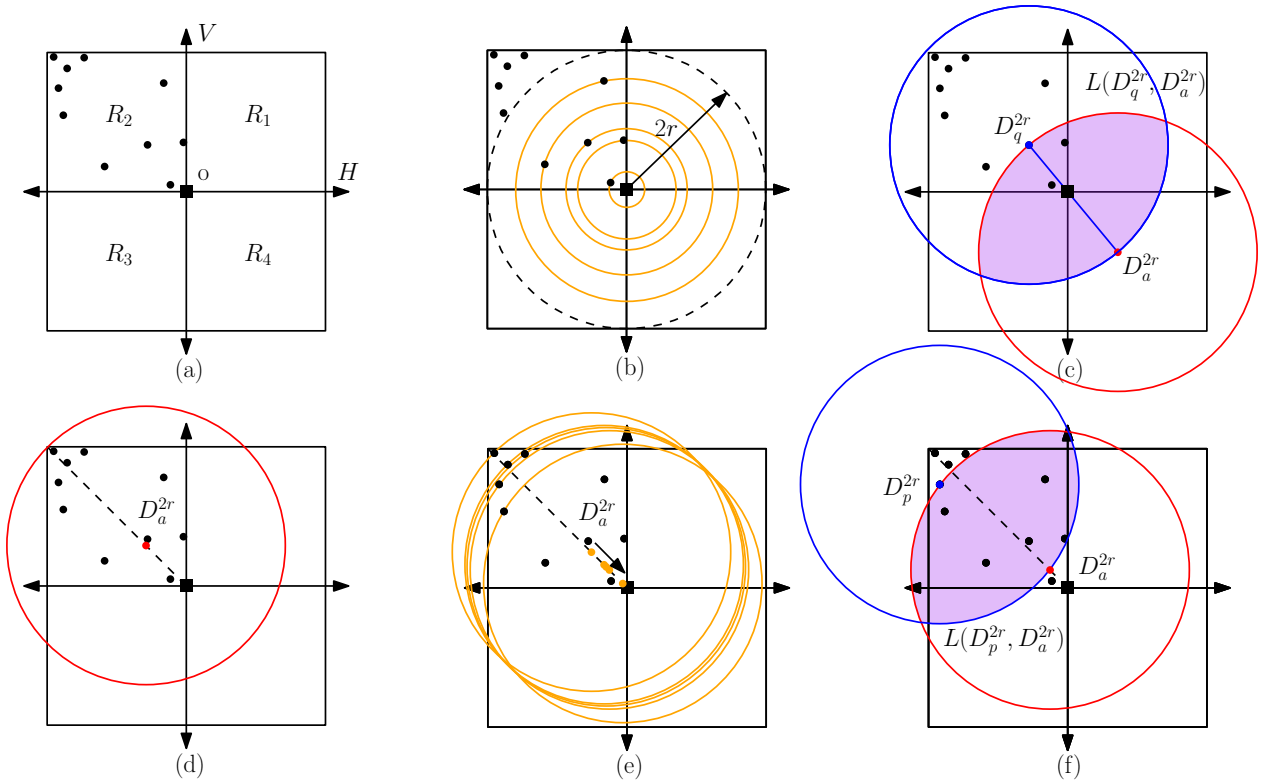

Figure 4.2: (a) The input square and disk centers. For simplicity, we only consider disks in R_2.(b) The set of disk centers which the first process finds. (c) An example of the generated lens from one of the disks. (d) The starting position of D_a^{2r} for the second process in region R_2. (e) The set of disk centers the second process finds. (f) An example of the generated lens from one of the disks.

the max clique M. The strategy of this proof is to show that the *solution set* F, that is, a region generated by D_c^r which contains the points of M, is a subset of the lens generated through Algorithm ***Square***. To do this, we will show that in both cases the disks which form the lens, namely one disk which is formed from D_c^r, and one disk which is artificially added, D_a^{2r}, are both safe choices for finding M. We begin this proof as follows.

Lemma 4.3.1. *Let S be the square generated by Algorithm* **Square** *and D_c^r be a disk with radius r and centre c. If $D_c^r \in M$, then the lens generated by $D_c^r \in S$ in Algorithm* **Square** *contains M.*

Proof. Recall that the maximum clique of S must include a disk from diagonal regions R_1 and R_3, or from R_2 and R_4. If our max clique exclusively involved two regions which are adjacent, then that clique would have already been recursively solved from points contained in one of the sets P_l, P_r, Q_t, or Q_b. Next, we consider two cases, depending on whether c is greater than a distance of $2r$ from o or not.

Case 1: c is within or equal to a distance of $2r$ from o: Observe that if c is in R_1, then a must be in R_3. This is similarly true for R_2 and R_4. Recall that c is assumed to be the farthest point in M from o. Any disk whose centre is in D_c^{2r} is adjacent to c, and so our choice of D^2r_c is sufficient. It remains to show the choice of D_a^{2r} is sufficient.

Without loss of generality, assume c is in R_2. Let J be the circular region centered at o with radius $|co|$ (Figure 4.3 (Left)). Since D_c^r is assumed to be the disk farthest from o in the solution, it must be that our solution set F is a subset of $D_c^{2r} \cap J$. We will now show $F \subseteq L(D_c^{2r}, D_a^{2r})$, justifying the choice of D_a^{2r}.

Assume for contradiction that there exists a disk D_k^r which is in F but not in $L(D_c^{2r}, D_a^{2r})$. If $k \notin D_c^{2r}$, then by definition D_k^r cannot be adjacent to D_c^r. Instead, consider $D_k^r \in (F \cap D_c^{2r}) \setminus D_a^{2r}$. Since $|co| \leq 2r$, D_a^{2r} will always at least be as big as J. Furthermore, since J and D_a^{2r} are defined by D_c^r and lie on the same line segment ca, it must be that D_a^{2r} contains J, and by extension, F.

Since D_k^r cannot exist, we arrive at a contradiction and thus $F \subseteq L(D_c^{2r}, D_a^{2r})$. This proves D_a^{2r} is a safe choice, and by extension, $L(D_c^{2r}, D_a^{2r})$ contains M.

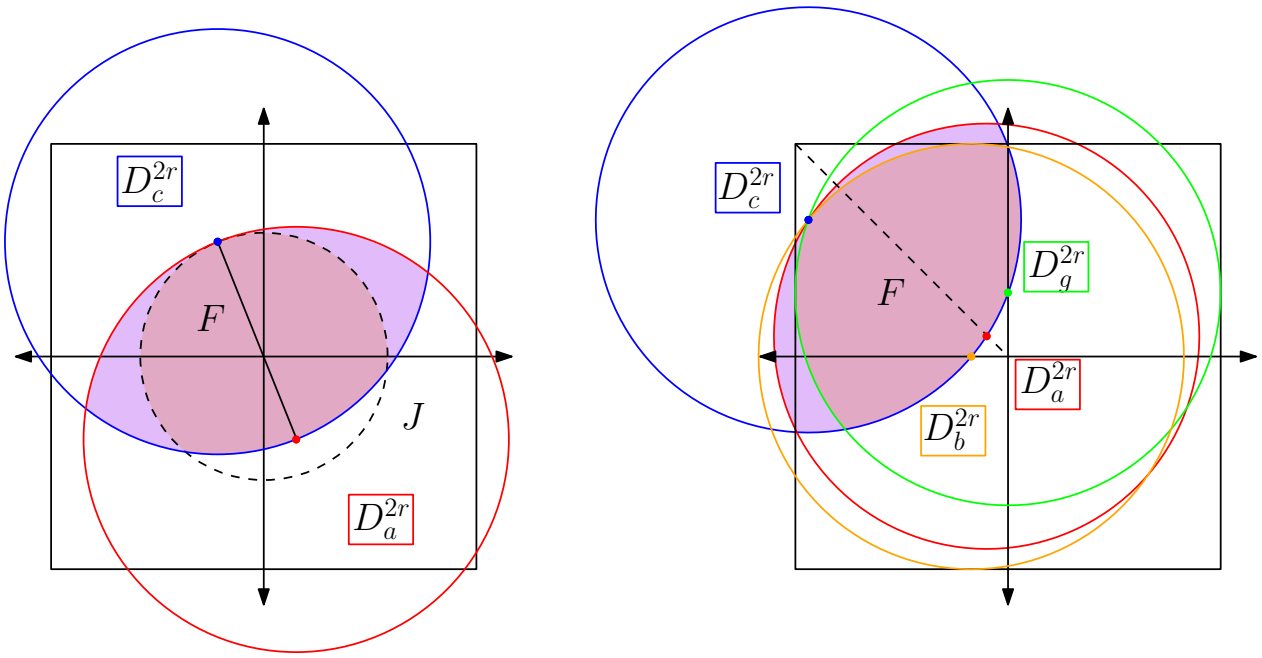

Figure 4.3: (Left) An illustration of Case 1. Here, the dashed circle represents J. (Right) An illustration of Case 2. Observe in this example there exists a portion of the lens in R_3 which lies outside F. Without D_g^{2r}, this portion would be incorrectly included in F.

Case 2: c is greater than a distance of $2r$ from o:

Without loss of generality, assume D_c^r resides in R_2. Here, we will identify our solution set F by first constructing two additional disks. Let D_g^{2r} be the disk whose center is positioned at $V \cap D_c^{2r}$. If there are two choices for the position of g, choose the one closer to o. We similarly define D_b^{2r} with its center at $H \cap D_c^{2r}$. Again, if there are two choices, we choose the one closer to o. For now, we assume that the solution set corresponds to $F = D_c^{2r} \cap D_g^{2r} \cap D_b^{2r}$ (Figure 4.3 (Right)). It is obvious that the choice of D_c^{2r} is safe as one of the disks to generate our lens. It then remains to show that the choice of D_a^{2r} is safe.

Because a, g and b lie on the boundary of D_c^{2r} and their disks all have the same radius, their intersection will be the two disks farthest apart. As D_a^{2r} lies between the other two, it must be that $F \subseteq D_a^{2r}$, and as an extension $F \subseteq L(D_c^{2r}, D_a^{2r})$. It must now be proven that our assumption of $F = D_c^{2r} \cap D_g^{2r} \cap D_b^{2r}$ is correct.

Clearly, all disks apart of our maximum clique must intersect D_c^r, and so all disk centers must be contained in D_c^{2r}. Observe that D_c^{2r} does not intersect R_4. By our previous observation, it must be then that there is at least one disk in R_1, and at least one disk in R_3, both of which intersect D_c^r. Furthermore, these disks from R_1 and R_3 must intersect each other. This imposes a restriction as to how far apart disks from these regions can be while still being apart of the maximum clique with D_c^r. For R_1, the closest a disk can be to R_3 is represented by D_g^{2r}. Similarly, D_b^{2r} represents the closest a disk can be positioned in R_3 to R_1 while still intersecting D_c^r. Therefore, any disk centre which does not intersect one of D_g^{2r} or D_b^{2r} cannot intersect any disk in their associated region, and thus cannot be apart of the clique we are trying to determine. So, the solution set must include the disks which intersect exactly D_g^{2r}, D_b^{2r}, and D_c^{2r}.

Since the solution set is correct, and the choice of D_a^{2r} is correct given the solution set, we can safely conclude that $L(D_c^{2r}, D_a^{2r})$ contains M. As all disks in S are handed by either of the above cases, for any guess of D_c^r, we can correctly generate the lens which would contain its maximum clique M.

□

This proof culminates in the following algorithm and theorem.

Theorem 4.3.2. *The maximum clique M of a unit disk graph G with a representation of n disks on the Euclidean plane can be calculated in $O(n^{2.5} \log n)$ time.*

Proof. Assume each disk is of radius r. Let P be the set of centers of the disks corresponding to the vertices of G. Then, we can construct the maximum clique of G by the following algorithm:

Algorithm *Unit Clique*

1. Rotate the plane so that no two points are on the same vertical or horizontal line.
2. Sort P by their x-coordinate and partition P into two disjoint sets, P_l and P_r, by a vertical line Z positioned at the median of P. Recursively find the max clique of P_l and P_r respectively.
3. Since M may contain disks from P_l and P_r, we merge P_l and P_r and keep the largest clique found by the following subroutine:
   (a) Create two additional vertical lines Z_l and Z_r at a distance $-2r$ and $+2r$ from Z and define a new set Q, the set of disks between Z_l and Z_r
   (b) Sort Q by their y-coordinate and partition Q into two disjoint sets, Q_t and Q_b, by a horizontal line H positioned at the median of Q. Recursively find the maxi clique of Q_t and Q_b respectively.
   (c) Since the max clique may contain parts of Q_t and Q_b, we merge Q_t and Q_b and keep the largest clique found by the following subroutine:
      i. Define a square S of size $4r$ centered at $V \cap H$
      ii. Return the maximum clique found in S by running Algorithm *Square*
4. Return the maximum clique M found over steps (a)–(c)

By the time analysis in Section 4.1, and the proof of correctness for Algorithm ***Square***, it must be that the time complexity of Algorithm ***Unit Clique*** is dominated by the subroutine in Algorithm ***Square***. Therefore Algorithm *Unit Clique* can calculate the maximum clique of a disk graph in $O(n^{2.5} \log n)$ time.

□

4.4 Summary

The primary bottleneck in the algorithm by [14] is the need to compare each disk against each other disk to identify which disk pair is the farthest apart in the maximum clique. Instead, using a divide-and-conquer algorithm, we inspect each element of the unit disk graph once. This was accomplished in two major ways. Firstly, the restrictive geometry found within the square provided the resources to develop a lens using only one assumed disk. This is because the lines Z and H are defined around medians of the unit disk graph and allow us to predict where the "best" lens involving a unit disk must be. Put another way, a lens generated from a disk D_c^r which does not follow the constructive methods found in Algorithm ***Square*** would be positioned in such a way so as to exclude a better maximum clique with D_c^r being the farthest disk from it. This ties in closely to the second major reason, which is the use of divide-and-conquer. By recursively partitioning at the median, the square restricts a given area to the largest possible space a clique could occupy around $V \cap H$. These ideas combine to provide a technique which need only guess one disk D_c^r at a time, as the guessing of a second disk is avoided by already determining what a good lens formed with D_c^r would look like.

The opportunity to improve upon the running time of maximum clique in a unit disk graph is a strong result in itself. More importantly, this algorithm provides more unique methods to solving disk graphs in general. This result additionally opens the potential of applying similar methods to more general disk graphs, or other variants of it. Hopefully, techniques such as divide-and-conquer, or more precise geometric utilization, will continue to provide advances to solving the long standing open problem posed by Ambühl and Wagner [3].

5 Hardness for Disk-Axis Graph

In this section we will introduce a way to orient and label any Hamiltonian cubic graph in what we call a *pair-oriented labelling*. This process employs additional restrictions to the typical co-2-subdivision technique. In this section, we will define and provide an algorithm to construct a pair-oriented labelling. Furthermore, we provide an application for this labelling via a reduction to Disk-Axis Clique. This is a known NP-hard problem, however NP-hardness was previously thought difficult to be proved using co-2-subdivision. Using pair-oriented labelling, we show that a careful ordering of the vertices allows for this problem to be solved effectively with co-2-subdivision.

5.1 Pair-Oriented Labelling

Let $G = (V, E)$ be a Hamiltonian cubic graph. For each edge $e \in E$, we assign e a label and an orientation. We denote the label of e as $L(e)$, where $L(e) \in \mathbb{N}$. Note that no two edges may share the same label. Furthermore, we denote the orientation of e as $O(e)$, where $O(e)$ is either an *incoming* or *outgoing* edge relative to some vertex. A vertex is defined as *satisfied* if the following is true: there are two edges, e and u, incident to v such that $O(e) = O(u)$ relative to v, and that $|L(e) - L(u)| = 1$. We say G has a *pair-oriented labelling* if there exists an orientation and labelling for each edge $e \in G$ such that each vertex $v \in G$ is satisfied. We denote $P(G)$ as a pair-oriented labelling for G, if it exists. Informally, $P(G)$ has the property that every vertex has two edges which have the same orientation and are consecutively labelled. Please refer to the illustration in Figure 5.1 (Left)-(Middle).

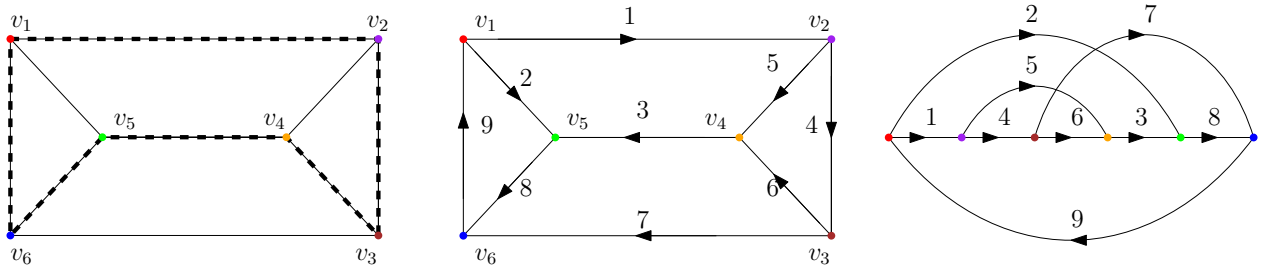

Figure 5.1: (Left) A Hamiltonian cubic graph. A Hamiltonian cycle h is highlighted by a dashed line. (Middle) The resulting pair-oriented labelling. (Right) An updated drawing of the graph.

We now show that for any Hamiltonian cubic graph G, there existing a pair-oriented labelling of G.

Theorem 5.1.1. *If G is a Hamiltonian cubic graph with, then a pair-oriented labelling, $P(G)$, exists and*

can be calculated in polynomial time.

Proof. This proof will follow the format of a proof by construction, showing that there is a set of steps for G which can always be taken satisfy to each vertex, and consequently, obtain $P(G)$.

Let $G = (V, E)$ be a Hamiltonian cubic graph with n vertices. Let $h = v_1e_1v_2e_2 \dots v_ne_n$ be a Hamiltonian cycle in G with $v_i \in V$, $e_i \in E$, and $1 \leq i \leq n$. If an edge $e \in h$, then we say e is a *Hamiltonian edge*. Otherwise, e is a *non-Hamiltonian edge*.

We first orient the edges of h so that they form a directed cycle. That is, after forming the directed cycle, each vertex of G has one incoming edge, one outgoing edge (both of which are Hamiltonian edges), and one unoriented, non-Hamiltonian edge. For each unoriented edge, $e = (v_i, v_j)$ with $i < j$, we orient e as an outgoing edge relative to v_i. Informally, this represents "combing" the non-Hamiltonian edges in one direction. To help visualize this, the graph is represented with its vertices placed along the horizontal axis (Figure 5.1 (Right)).

Observe that orienting the edges in this manner allows each vertex to have at least two edges of the same orientation. This meets one of the conditions of a satisfied vertex. Then it remains to prove that the edges can be labelled correctly provided this orientation. To do this, we will label the edges of G by an iterative construction. This process is described below.

Let v_k be the vertex with the smallest index k which has an unlabelled outgoing non-Hamiltonian edge. We define the *maximal bridge sequence* of v_k, denoted S_k, as the sequence of edges $e_1e_2 \dots e_q$. Note that in the maximal bridge sequence, edges do not need to follow their given orientation. S_k has the following properties:

- S_k represents a path in G which starts at e_1 and ends at e_q. Here, q is an odd integer.
- The vertex which joins e_1 and e_2 is v_k
- For all $e_i \in S_k$, if i is odd, then e_i is a Hamiltonian edge. If i is even, then e_i is a non-Hamiltonian edge.
- Let $e_ie_{i+1}e_{i+2}$ be a sub-sequence of S_k where i is odd with $1 \leq i \leq q-2$. Let v be the vertex which joins e_i and e_{i+1}, and u be the vertex which joins e_{i+1} and e_{i+2}. Then we have $O(e_i) = O(e_{i+1})$ relative to v, and $O(e_{i+1}) = O(e_{i+2})$ relative to u. Figure 5.2 (Left) illustrates a of a sub-sequence of S_k.

Once the edge sequence for S_k has been found, we can label each edge in S_k, and then repeat this process for the next v_k until there are no more valid choices for v_k.

We now turn to labelling the edges of S_k. Let $a \in \mathbb{N}$ be the largest label given so far in G. If no label is given yet, $a = 0$. By labelling the edges of $S_k = e_1e_2 \dots e_q$ as $a+1$, $a+2$, $\dots$ $a+q$, each vertex which has two edges that appear in S_k will be satisfied. It remains to show that the iterative construction of maximal bridge sequences will satisfy all vertices in G.

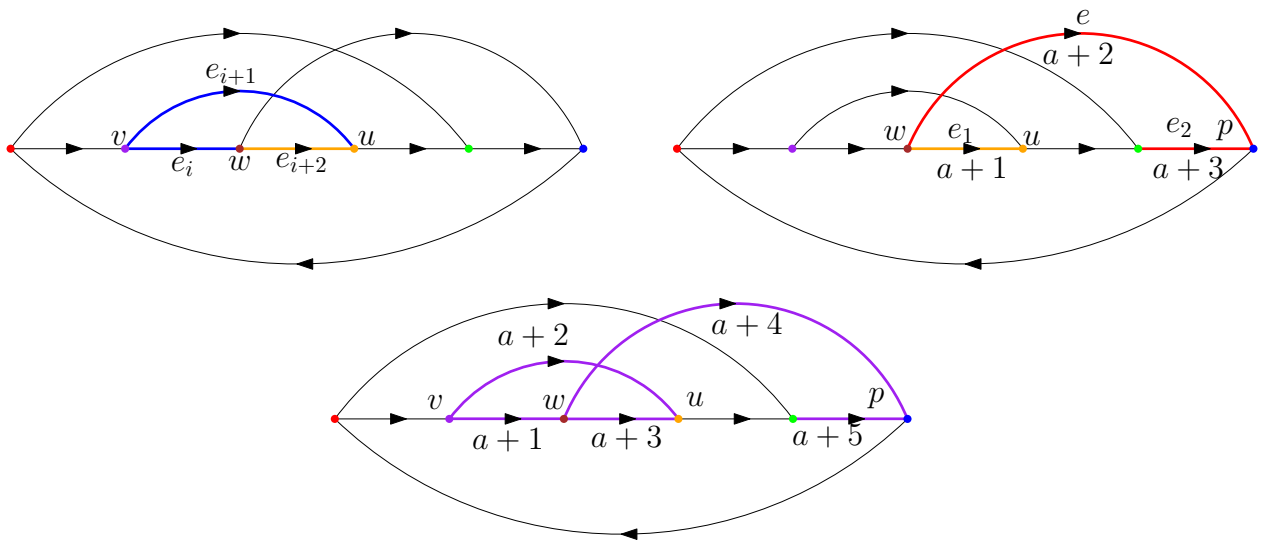

Figure 5.2: (Left) An example sub-sequence found in some maximal bridge sequence. (Right) The labelling for a sequences of edges found by the alternative labelling process A. (Bottom) The maximal bridge sequence from u. Observe that u, w, v, p are all satisfied.

To accomplish this, consider an alternative labelling process A, described as follows. Let e be a non-Hamiltonian edge with endpoints w and p. Since each vertex has at least one incoming and outgoing edge due to the directed cycle formed with h, it must be that $O(e)$ is one of the following. Either e is the second income edge for w and the second outgoing edge for p, or e is the second outgoing edge for w and the second incoming edge for p. Let e_1 be the Hamiltonian edge with endpoint w such that $O(e) = O(e_1)$ relative to w. Similarly, let e_2 be the Hamiltonian edge with endpoint p such that $O(e) = O(e_2)$ relative to p. Then labelling the sequence e_1, e, e_2 as $a+1$, $a+2$, $a+3$ satisfies both w and p. Figure 5.2 (Right) illustrates an instance of A.

Recall that because G is a cubic graph, it has the relationship $|E| = \frac{3}{2}|V|$. Then there are $|h| = |V| = |E| - \frac{|V|}{2}$ edges in h, and thus $\frac{1}{2}|V|$ edges not in h. If we allow edges to have more than one label, then the labelling scheme e_1, e, e_2 as $a+1$, $a+2$, $a+3$ can be independently replicated $\frac{1}{2}|V|$ times, once for each non-Hamiltonian edge e. Therefore, the process A can satisfy all vertices in G if edges are allowed to have multiple labels. We will now take A and adjust it so as to preserve its correctness, while removing the assumption that edges can have two different labels. This adjustment will result in iterative finding and labelling of maximal bridge sequences.

Assume we use A to label all edges in G accordingly. Observe that any edge in G which is labelled twice this way would be an interior edge for some maximal bridge sequence S_k. Then, the number of sequences created using A will never be greater then the number of maximal bridge sequences in G. Then for two labelled sequences created by A, if both sequences share an edge, then the union of the sequences may be relabelled as $a+1$, $a+2$, $a+3$, $a+4$, $a+5$. For example, in Figure 5.2, the orange edge acts as the source to merge the blue sequence (Left) with the red sequence (Right) to form the purple sequence (Bottom).

Additionally, the choice of v_k means each maximal bridge sequence starts with a non-interior edge.

What this means is one can merge and relabel sequences from A so that no edge has two labels and that each vertex which appears in these sequences remains satisfied. By repeatedly merging sequences from A which share an edge until no edge is labelled twice, the resulting structure can be equivalently described by some number of maximal bridge sequences. Therefore, with at most $\frac{1}{2}|V|$ maximal bridge sequences, it is possible to label all edges in G by an iterative construction of maximal bridge sequences. Since all vertices are satisfied after finding all maximal bridge sequences, it is trivial to label the remaining edges. □

This proof gives rise to an algorithm which can be used to find $P(G)$. Assuming the Hamiltonian cycle of G is given, we can construct $P(G)$ in polynomial time. Please refer to Fig. 5.3 for an illustration.

Algorithm *Pair Orient*

1. Take as input a cubic graph G with a Hamiltonian cycle h and order the vertices appropriately (Fig. 5.3(a))
2. Form a directed cycle from the edges in h (Fig. 5.3(b))
3. For each non-Hamiltonian edge $e = (v_i, v_k)$ with $i < k$, let $O(e) =$ outgoing relative to v_i (Fig. 5.3(c))
4. While there is a vertex with an unlabelled, outgoing non-Hamiltonian edge: (Fig. 5.3(d)-(g))
   (a) Identify the vertex with the smallest such index v_k
   (b) Calculate its maximal bridge sequence
   (c) Label the edges in the sequence starting with the next available label
5. Label the remaining unlabelled edges arbitrarily (Fig. 5.3(h))

5.2 Relationship to Co-2-Subdivision

The purpose of pair-oriented labelling is to use the labelling and orientation calculated for the edges, and transform that into a co-2-subdivision graph with additional information.

We will first describe the steps of transformation. Let G be a Hamiltonian cubic graph and $P(G)$ be the pair-oriented labelling of G. For each edge $e = (u, v)$ with orientation $O(e)$ and label $L(e)$, define two additional vertices, $x_{L(e)}$ and $y_{L(e)}$. Assume $O(e)$ is an outgoing edge relative to one of its endpoints u. We then perform a 2-subdivision on e by replacing the edge (u, v) with the path $u, y_{L(e)}, x_{L(e)}, v$. Recall $Sub_k(G)$ is the k-subdivision of G. By repeating this process for all edges, we transform an instance of $P(G)$ into a special instance of $Sub_2(G)$, called $PairSub(G)$. The vertices added through subdivision encode additional information by their labelling. This allows us to find $\overline{PairSub(G)}$, the special co-2-subdivision of $P(G)$. Figure 5.4 visualizes this process.

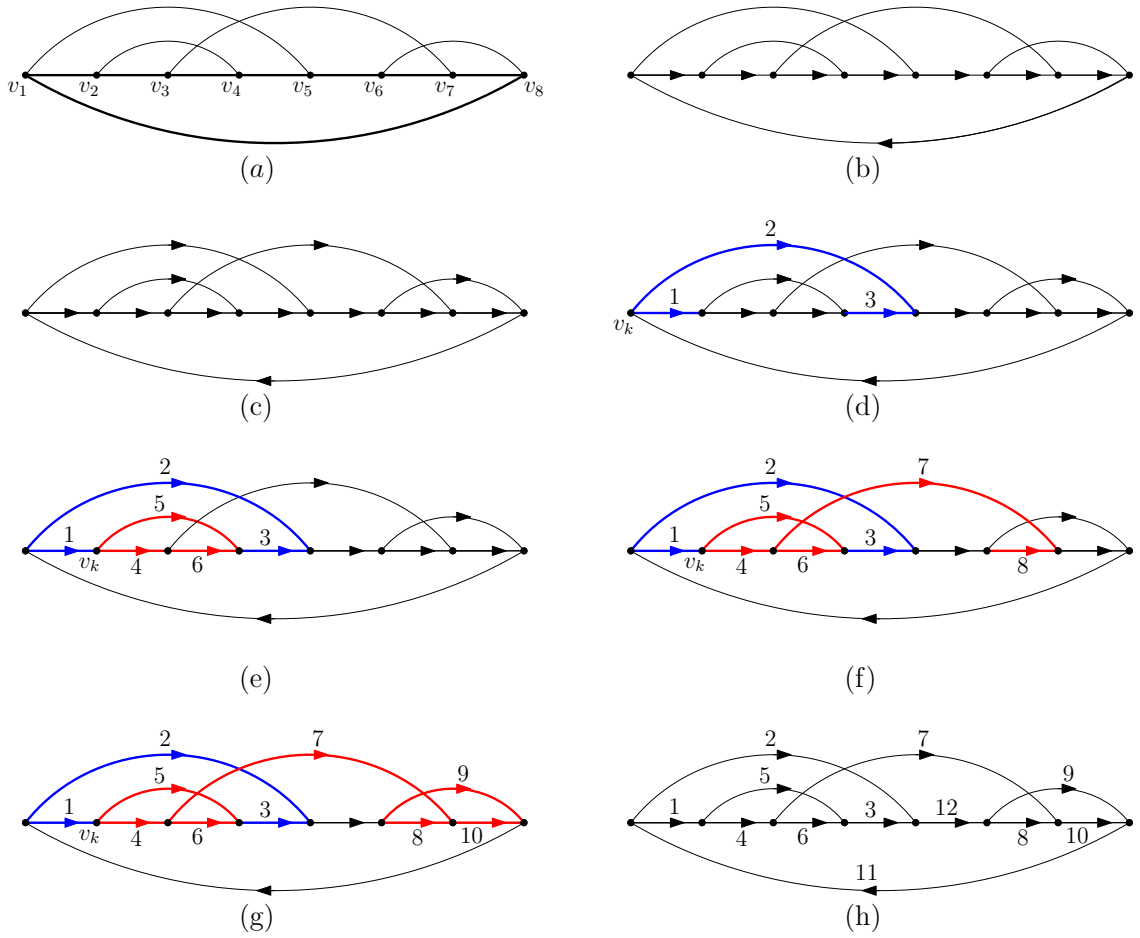

Figure 5.3: (a) The input cubic graph. (b)-(c) Orienting the edges. (d)-(g) The creation of the maximal bridge sequences and their labelling. (h)The computed $P(G)$, with the last two labels added.

The Pair-Orient Max Clique problem is defined as follows. Given a Hamiltonian cubic graph G where one of the Hamiltonian cycles is given, find a set of vertices that define the maximum clique of $\overline{PairSub(G)}$. Note that the problem of finding the max independent set in a cubic graph is NP-complete even when the Hamiltonian cycle is given as input [22]. Therefore, Pair-Orient Max Clique is NP-hard.

Given the graph $\overline{PairSub(G)}$, there are some useful properties about its structure that will be relevant in using Pair-Orient Max Clique. Recall that for graphs involving subdivisions, we refer to the vertices introduced in this process as *division vertices*. All other vertices are referred to as *original vertices*.

- **P1**: Each original vertex is adjacent to all but exactly three division vertices. These vertices will be in one of two forms. Without loss of generality, these vertices will take the form x_i, x_{i+1} and y_j, with $i \neq j$ and $(i+1) \neq j$.

- **P2**: Each division vertex is adjacent to all but exactly one original vertex and one division vertex. The division vertex it is not adjacent to will have the same subscript on its label. That is, if x_i is a division vertex, then it is not adjacent to y_i.

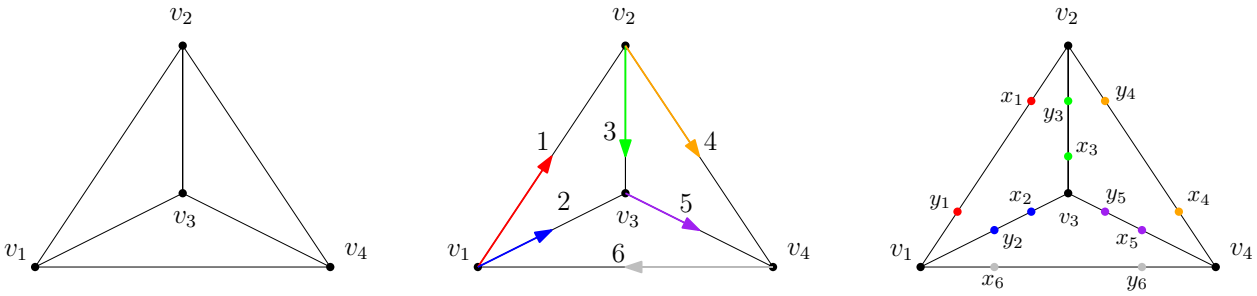

Figure 5.4: (Left) The input Hamiltonian cubic graph G. (Right) The calculated $P(G)$. (Bottom) The calculated $PairSub(G)$.

These properties define the structure found in $\overline{PairSub(G)}$, and will be used in subsequent sections.

5.3 Application to Unit Disks and Axis Aligned Rectangles

In this section we will use pair-oriented labelling to show that the problem Disk-Axis Clique is NP-hard. We define the problem Disk-Axis Clique as follows. Take as input a set of axis aligned rectangles R in the plane, and a set of unit disks D in the plane. Let G be the intersection graph of the set $\{R \cup D\}$. The goal is to find a set of vertices in G which represent a maximum clique in G. We will show that Disk-Axis Clique is NP-hard by performing a reduction from an instance of the Pair-Orient Max Clique problem.

It is already known that Disk-Axis Clique is NP-Hard, as shown in [6]. Therefore, this section aims to accomplish two goals. First, to provide an NP-hardness proof using a reduction from Pair-Orient Max Clique as a proof of concept for the technique. Second, to act as a response to the perceived difficulty in showing Disk-Axis Clique is NP-hard using traditional co-2-subdivision methods. In particular, the reduction from Bonnet et al. in [6] does not use a co-2-subdivision approach, as the authors believe it is difficult to implement for this problem.

We will prove the following theorem:

Theorem 5.3.1. *Finding the maximum clique in the intersection graph of unit disks and axis-aligned rectangles is NP-hard.*

Proof. Let S be the intersection graph of unit disks and axis-aligned rectangles. Let $g = (V, E)$ be a Hamiltonian cubic graph and $G = \overline{PairSub(g)}$. We will employ a polynomial time transformation from an instance of G to an instance of S. Note that this transformation holds similar structure to the one shown in [6].

Let o be the origin of the Cartesian plane for which S exists in. Let $d = |E| + 1$, where E is the edge set of g. We define four rays, x_d, x_0, y_d, and y_0, which all start at o and pass through the positive x-axis, positive y-axis, negative x-axis, and negative y-axis respectively. For each subdivision vertex in G with label x_i, we draw a line passing through the points $(i, 0)$, and $(0, d - i)$. Similarly, each subdivision vertex with label y_i passes through points $(-i, 0)$ and $(0, i - d)$. These arrangements of lines represent the set of division vertices

from G. Observe that these sets of lines are arranged as a *sail* arrangement [19]. In this way, the set of lines representing vertices with label x_i are collectively referred to as X. We similarly define Y as the set of lines representing the division vertices with label y_i. Note that these lines will later be interpreted as half-planes, and then eventually very large disks. Figure 5.5 displays the construction of $X \cup Y$ for an instance of S up to this point.

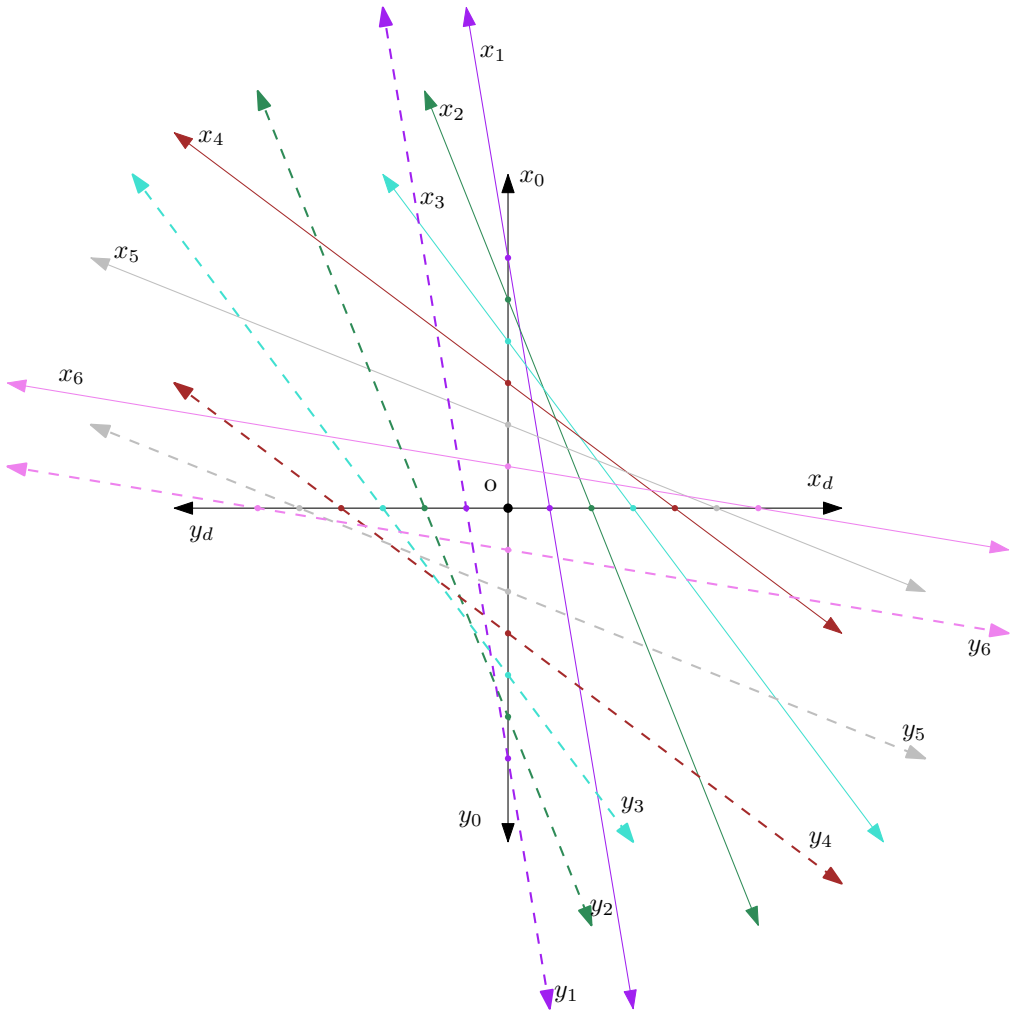

Figure 5.5: Representation of the division vertices from G as lines in S. The set X contains the bold lines, while the set Y contains the dashed lines. In this example, $d = 7$.

We now insert the axis-aligned rectangles into S, representative of the original vertices in G. For each original vertex v, let v_x and v_y be points placed in quadrant 1 and quadrant 3 respectively. These two points form opposing corners for the axis-aligned rectangle which represents v. The goal is to place v_x and v_y carefully so that the resulting rectangle will preserve its adjacencies from G to S.

Consider the partition of quadrant 1 generated by X. Each of the resulting bounded regions can be described by the lines which define it. For example, in Figure 5.5, the region closest to o can be identified as the region bounded by x_7, x_0, x_1 and x_6. Each region can be uniquely identified this way. We can similarly

define regions in quadrant 3 partitioned by Y. Then, without loss of generality, using property **P1**, for an original vertex v, we will place v_x in the region bounded by x_{i-1}, x_i, x_{i+1}, x_{i+2}, $1 \leq i \leq d-2$ and v_y in the region bounded by y_{j-1}, y_j, y_{j+1}, $1 \leq j \leq d-1$. Observe that by the construction of v that v_x is bounded by a quadrilateral, and v_y is bounded by a triangle. The placement of these vertices defines the resulting axis-aligned rectangle associated with each original vertex from G. We may now consider the lines representing division vertices as half-planes such that $X \cap o = \emptyset$ and $Y \cap o = \emptyset$. That is, the half-planes form X and the half-planes form Y point away from the center. Finally, these half-planes can be approximated by very large disks, completing the construction of S. A completed construction, and the associated graph $PairSub(G)$ can be seen in Figure 5.6.

It now remains to justify that this construction preserves the instance of G as an instance of S. To do so, we will shows that properties **P1** and **P2** are preserved in the construction.

Preserving P1: First observe that because the placement of v_x and v_y are bounded by quadrant 1 and 3 respectively, the resulting axis-aligned rectangle must intersect o. Thus, all axis-aligned rectangles in S will intersect o, and so the intersections of original vertices between one another are preserved.

Without loss of generality, it must be that the axis-aligned rectangle v intersects all but three division vertices of the form x_i, x_{i+1} and y_j with $i \neq j$ and $i+1 \neq j$. Recall that by the construction of v that v_x is bounded by a quadrilateral, and v_y is bounded by a triangle. Consider the arrangement of X and the points which defined its initial construction. We have that the order in which these lines intersect the x-axis (starting from closest to o) is x_1, x_2, ..., $x_{|E|}$. Therefore, the ray $(v_x, 270°)$ will intersect at least the lines x_a, x_{a+1}, ..., $x_{|E|}$ for some $1 \leq a \leq |E|$. Similarly, the ray $(v_x, 180°)$ will intersect at least the lines x_b, x_{b-1}, ..., x_1 for some $1 \leq b \leq a$. These each represent a sequence of lines which v must intersect. It then remains to show that x_a and x_b are x_{i+2} and x_{i-1} respectively (Fig. 5.7(Left)).

Notice $b < i < i+1 < a$ since if we instead assume that $i < i+1 \leq b < a$ or $b < a \leq i < i+1$, then v_x would lie outside the quadrilateral defined by x_{i-1}, x_i, x_{i+1}, x_{i+2}. This contradicts the construction process. Therefore, v_x does not intersect x_i or x_{i+1}. Then by the position of v_x, it must be that $x_a = x_{i+2}$ and $x_b = x_{i-1}$. A similar deduction can be made for the arrangement of Y. Therefore v will intersect all lines (disks) and rectangles except for exactly x_i, x_{i+1} and y_j (Fig. 5.7(Right)). This concludes the presence of **P1**.

Preserving P2: For the line x_i, its associated line y_i is the only parallel line in S, and thus x_i will intersect all division vertices except for y_i. When viewing these lines as very large disks, we may simply choose sizes of disks which address all relevant intersections. For each line in S, it must intersect all but one rectangle. This relationship is already implied through **P1**. This concludes the presence of **P2**.

By the above construction and justification, we have shown that finding the maximum clique in the intersection graph of unit disks and axis-aligned rectangles is NP-hard by showing any instance of the Pair-Orient Max Clique problem can be reduced in polynomial time into an instance of the Disk-Axis Clique problem. □

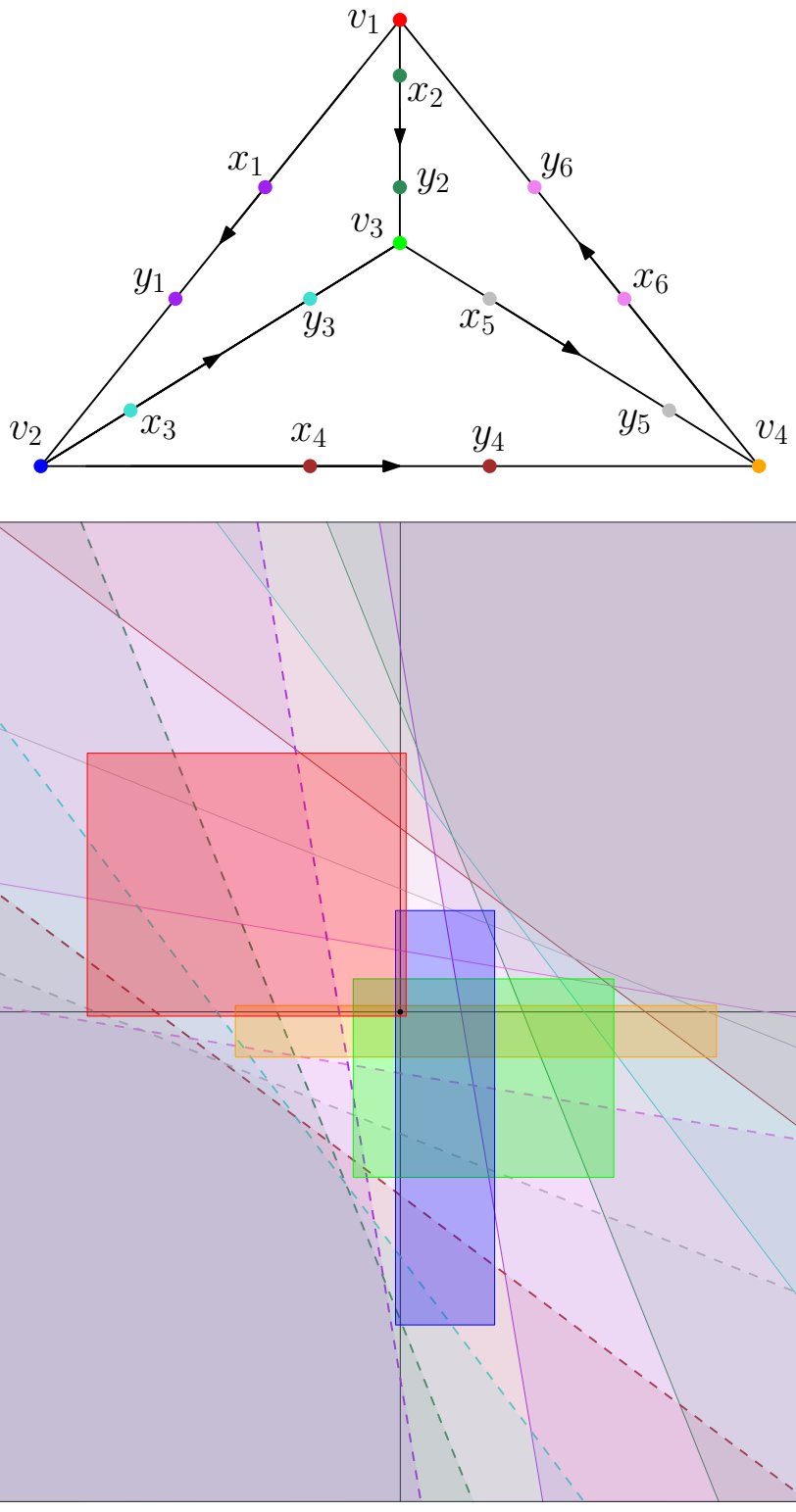

Figure 5.6: (Top) The graph $PairSub(G)$ with labelled vertices. (Bottom) The completed transformation to an instance of Disk-Axis Clique. Observe all four rectangles intersect the origin.

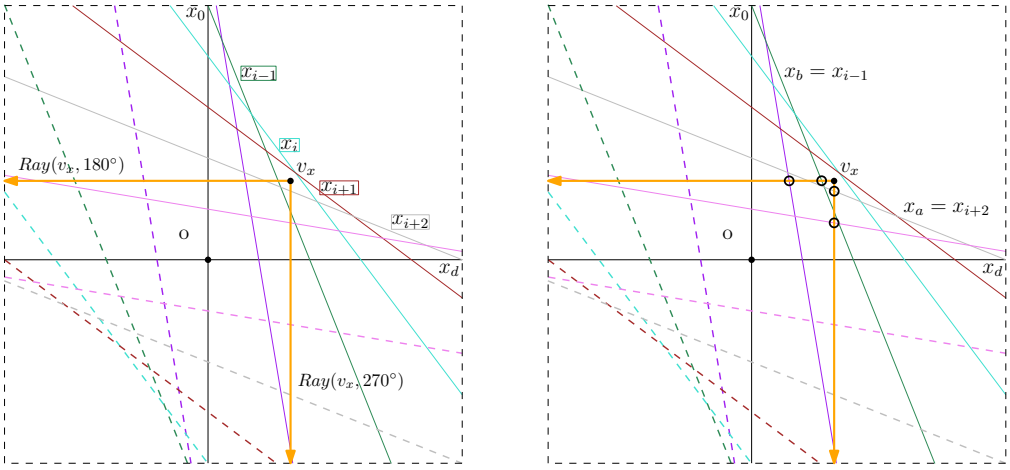

Figure 5.7: (Left) A visualization of the two rays formed at v_x. (Right) The highlighted intersections and there respective order. In this example, $a = 5$ and $b = 2$.

5.4 Summary

The completion of this proof naturally extends itself to other variations of this problem being NP-hard. It is easy to see how treating the disks as half-planes instead leads to an adjacent problem, which is also NP-hard. Other straight-forward modifications which lend themselves to being NP-hard problems would be altering the axis-aligned rectangles to triangles, where each axis-aligned rectangle is cut in half along the diagonal defined by v_x and v_y so as to preserve the half of the triangle which intersects o. These modifications may also be combined to show the graph of half-planes and triangles is also NP-hard. Other such adjacent problems may exist. In each modification, the origin of the Cartesian plane is used as a kind of anchor point for a geometric object, while the arrangement of lines, which bears similarity to mirrored sail arrangements [19], provides an effective way to permute the various intersections.

Recall that Bonnet et al. showed Disk-Axis Clique to be NP-hard in [6] by a reduction from MIPA (Max Interval Permutation Avoidance), a problem designed to solve Disk-Axis Clique. The authors state that the co-2-subdivision approach is difficult in part because, traditionally, each rectangle will need to avoid three arbitrary half-planes. What pair-oriented labelling enables is more control over where the third arbitrary half-plane must be positioned, namely that it must be consecutive to one of the other half-plane which need to be avoided. More generally, this displays that we can exploit the adjacency properties given in $PairSub(G)$ to simplify some maximum clique problems. In the case of Disk-Axis Clique, the sequential nature of two of these half-planes needing to be avoided meant we could in a sense treat the pair as one larger, awkward line segment to be avoided. These advantages show that encoding additional information into a traditional co-2-subdivision approach allows for potentially more graphical intersection problems to be solved.

6 Outer Triangle Graph Is NP-Hard

In this section, we will utilize pair-oriented labelling to demonstrate the hardness for calculating the maximum clique in an *outer triangle graph*. As in Section 5, we will perform a reduction from Pair-Orient Max Clique, this time to an instance of Outer Triangle Clique, to prove that such a problem is NP-hard. Unlike Disk-Axis Clique, the outer triangle graph presents a novel intersection graph for which to uniquely apply the advantages of pair-oriented labelling.

6.1 Max Clique in the Outer Triangle Graph is NP-Hard

Let H be a line and T be a set of triangles in the plane. Consider a triangle $t = \triangle abc$. Without loss of generality, if side $b \cap H = b$, that is, b is some segment of H, then we say t is *grounded*. If T is a set of grounded triangles such that all triangles lie on one side of H, then the intersection graph of T is known as a *grounded triangle graph*. Consider instead two parallel lines H and J, and a set of triangles T. Assume all triangles of T are grounded to either H or J. If all triangles in T lie in between H and J such that they do not intersect the opposing horizontal line, then the resulting intersection graph is known as an *outer triangle graph* (Fig. 6.1). It is worth noting that this arrangement is similar to the *triangle graph* [34] and *simple-triangle graph* [35], and bears resemblance to the trapezoid representation of a *trapezoid graph* [20]. The primary difference of the outer triangle graph is that it does not intersect both horizontal lines.

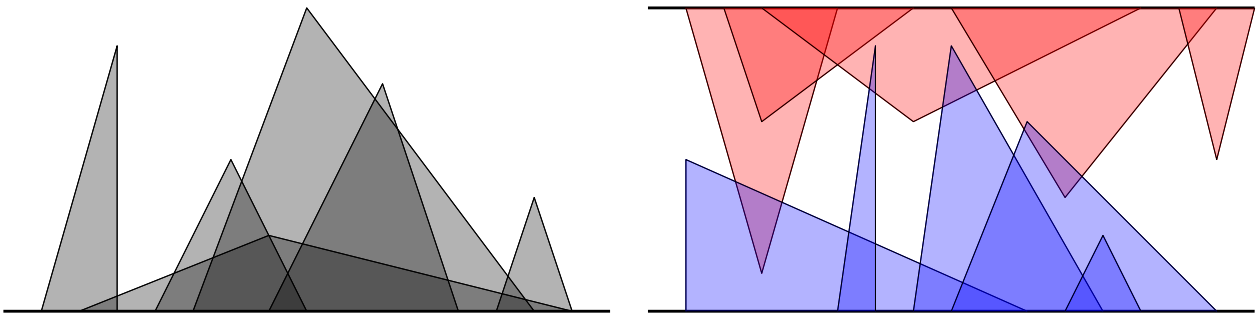

Figure 6.1: (Left) A representation for a grounded triangle graph. (Right) A representation for an outer triangle graph. The triangles are coloured according to the line they are attached to.

In this section we will use pair-oriented labelling to show that the problem Outer Triangle Clique is NP-hard. We define the problem Outer Triangle Clique as follows. Take as input an outer triangle graph G. The goal is to find a set of vertices in G which represent a maximum clique in G.

Theorem 6.1.1. *Calculating the maximum clique of an outer triangle graph is NP-hard.*

Proof. In this proof we will perform a reduction from an instance of Pair-Orient Max Clique to an instance S of Outer Triangle Clique. Let $g = (V, E)$ be a Hamiltonian cubic graph with n original vertices, $k = |E|$, and $G = \overline{PairSub(g)}$. The goal of this proof is to provide a constructive process of turning G into an instance of S.

Let o be the origin of the Cartesian plane. We define a regular polygon P with $4k$ sides centered around o. Since the number of sides of P is a multiple of 4, we rotate the polygon such that each quadrant of the plane has exactly k sides. These sides are then labelled as s_p^q, where q denotes the quadrant, and p denotes its position in a counterclockwise order. Then the sides of P are labelled as $s_1^1, s_2^1, \ldots, s_{k-1}^1, s_k^1, s_1^2, s_2^2, \ldots, s_{k-1}^4, s_k^4$. Note that P represents a structure (Fig. 6.2) which will be utilized throughout the construction of S, but will not be present in the final graph– it is used as a reference tool for other parts of the construction.

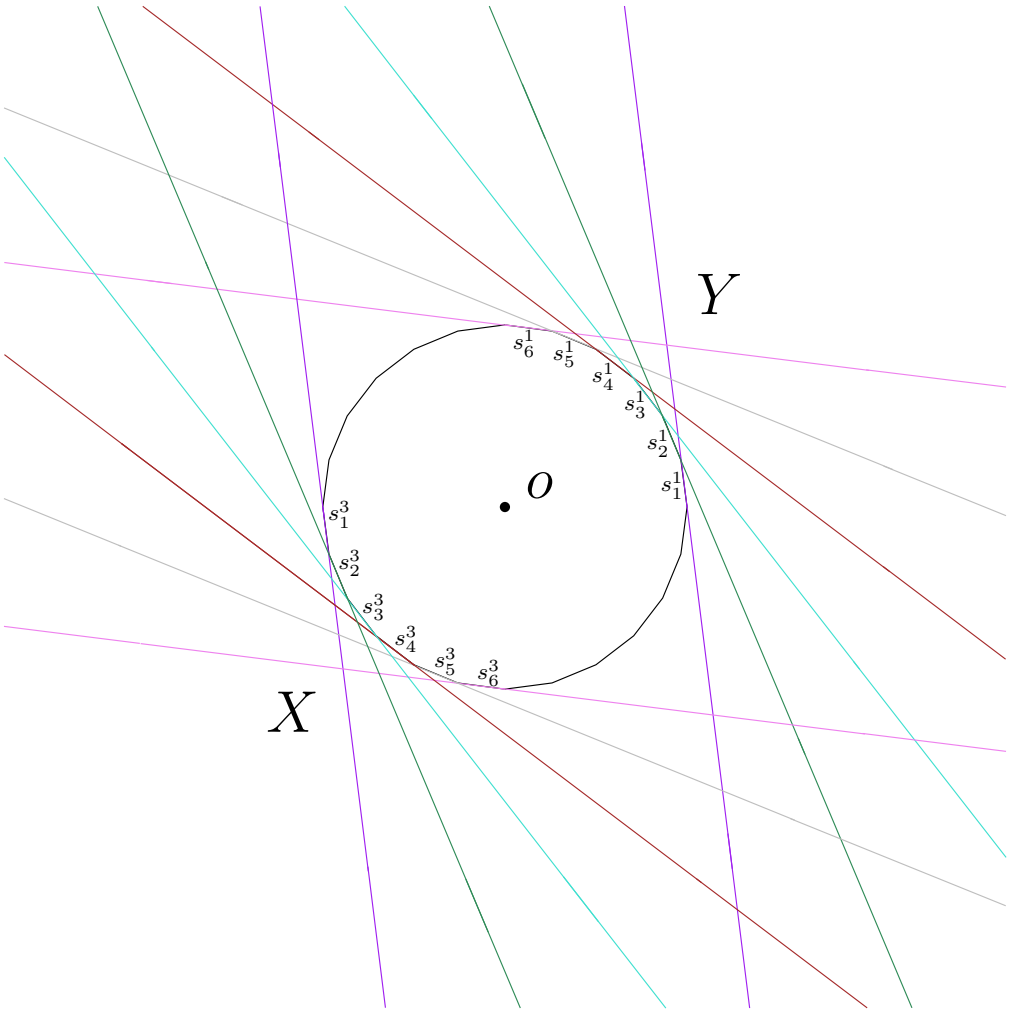

Figure 6.2: The placement of division vertices as lines, and the regular polygon which defines their placement. In this example, g has four vertices and six edges. Only the relevant sides of P are labelled.

As in the construction of Disk-Axis Clique, we partition the division vertices of G into the sets X and Y, for division vertices with label x_i and y_i, $1 \leq i \leq k$, respectively. Observe that $|X| = |Y| = k$. For each

division vertex $x_i \in X$, we insert a line representative of x_i that is determined by side s_i^3. Similarly, each element y_i of Y is represented as a line inserted over top of side s_i^1. Figure 6.2 details construction up until this point. Note that these lines will eventually be converted into triangles.

We now turn to the inserting of original vertices into S. Here, each vertex will be represented initially as a *dual ray*, that is, two rays (half-lines) which share the same source vertex. Similarly to the lines representing division vertices, these dual rays will eventually be converted into triangles. All introduced rays will be parallel to some existing line in S. In addition, each ray will be pointed in the direction towards o (Fig. 6.4 (Right)). That is, let r be a ray with endpoint e such that r is parallel to some line in S, and c be the point on r which is closest to o. Then we have that $|eo| > |co|$. Given the restrictions of the rays in this construction, we describe a dual ray as p_b^a, with p being the shared endpoint, and a, b being the lines which each ray is parallel to. Once the construction is complete, these dual rays, as well as the lines representing division vertices, will be re-expressed as grounded triangles (Fig. 6.3), representing an outer triangle graph. In principle, the size of these triangles can be any width and height so long as the transformation preserves adjacencies.

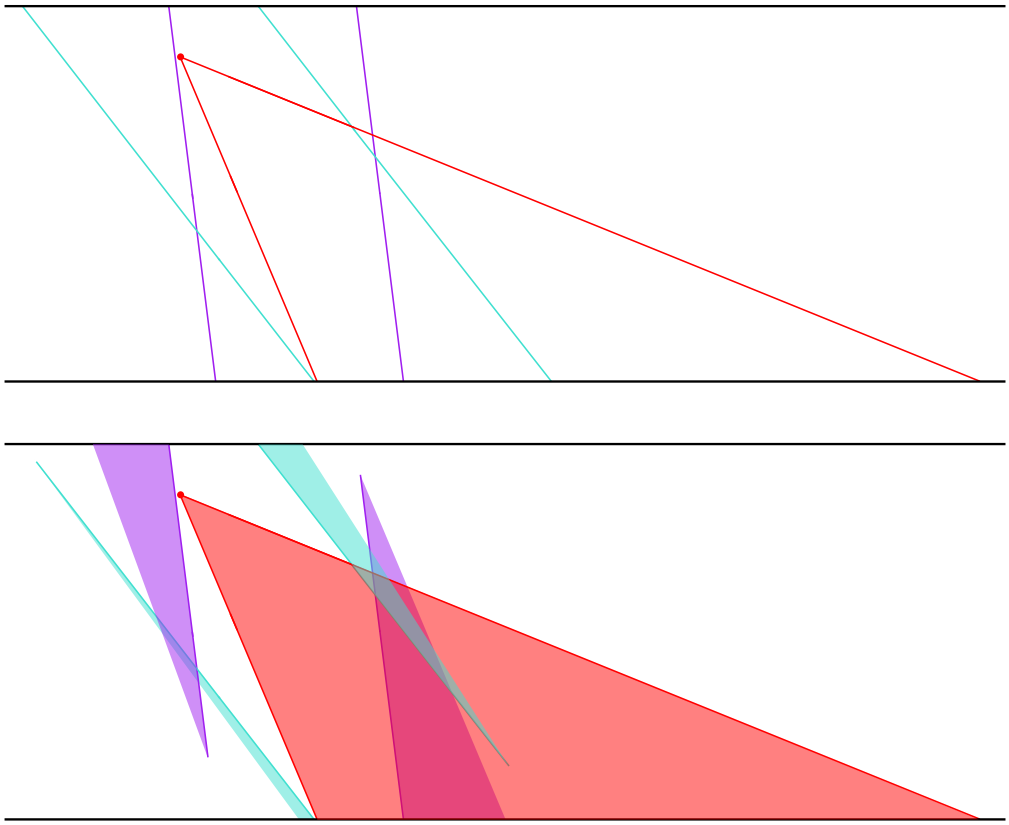

Figure 6.3: (Top) A greatly simplified instance of Outer Triangle Clique. Here, we have one dual ray (red) and four lines, representing an original vertex and four division vertices respectively. The black lines which will ground all of the triangles are placed close for more visual clarity. (Bottom) The conversion of each line and dual ray into grounded triangles.

Next, we define the placement for each dual ray. Without loss of generality, let v be an original vertex

from G which must intersect all but the three division vertices x_i, x_{i+1}, and y_j (property **P1** from Section 5.2). Consider the intersection points of x_i and y_j, and x_{i+1} and y_j, called I_1 and I_2 respectively. It must be that one of I_1 or I_2 is closer to o. Assume that I_1 is closer to o. As I_1 is the intersection of a line from X and Y, it must be that I_1 is in quadrant 2 or quadrant 4. Assume that I_1 is in quadrant 2 (Fig. 6.4 (Left)). We define a new point I such that I lies on the line segment I_1o, and that I is close to I_1 so as to not cross any line from $X \cup Y$. We then form the dual ray $I_{y_j}^{x_{i+1}}$, which represents v. This process is repeated then for each original vertex in G. Refer to Figure 6.4 for a visual reference in creating a dual ray.

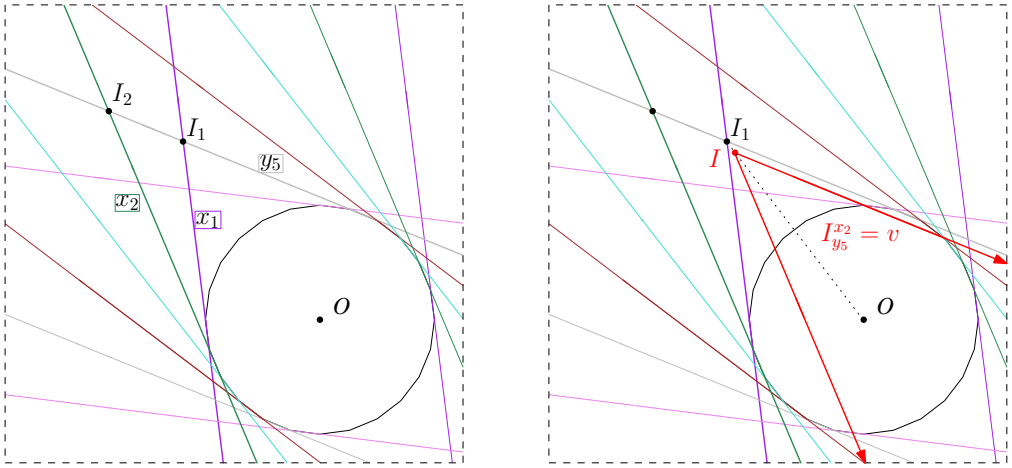

Figure 6.4: (Left) The relevant variables for creating the dual ray $I_{y_5}^{x_2}$. Here, the original vertex v which we want to represent in S has the three non-adjacent vertices x_1, x_2, and y_5. (Right) The construction of the dual ray for v (red). Observe that I lies on the dotted segment between I_1 and o.

Once all vertices of G are properly represented, the polygon P is removed from the plane. We now finalize the construction of S. First, define two horizontal lines T and B, such that T is placed arbitrarily above o so that all intersections in S occur below T. We similarly place B so that all intersections in S occur above B. All triangles will be grounded to one of these two lines. This placement of T and B allow all intersections in S happen between them. Additionally, the addition of T and B will cut all lines and rays so that all remaining elements of S are between T and B. This transforms the lines into line segments, and the dual rays into dual line segments.

Finally, for each line segment in $X \cup Y$, we may artificially treat them as very thin, grounded triangles by trimming the length from one endpoint, and then grounding the triangle on the opposite horizontal line (Fig. 6.3). It does not matter which line these thin triangle are grounded to. For each dual line segment with segments s_1 and s_2, without loss of generality, we can then form the grounded triangle $\triangle s_1 s_2 b$, where b is the region on B between s_1 and s_2. An example of a completed construction, and the associated graph $PairSub(g)$ can be seen in Figure 6.5.

It now remains to justify that this construction preserves G as an instance of S. To do so, we will show that properties **P1** and **P2** are preserved in the construction as stated in Section 5.2.

Preserving P1: Let v be a grounded triangle in S which represents some original vertex in G. Without

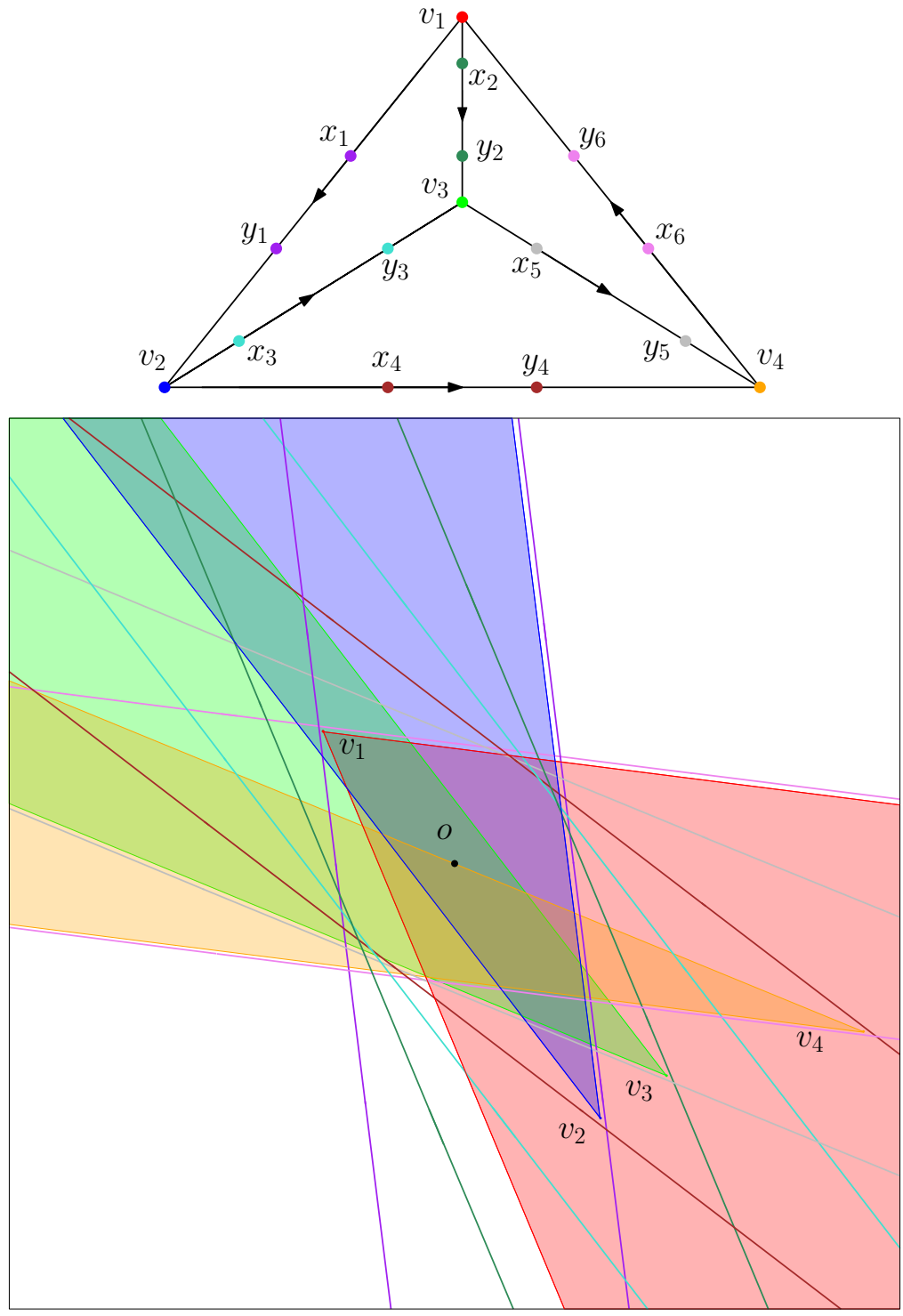

Figure 6.5: (Top) The input graph, shown as $PairSub(g)$. (Bottom) The resulting loos triangle graph. Observe that v_2, v_3, v_4 are all grounded to T, and v_1 is grounded to B. Each thin grounded triangle is represented by a thick line to reduce visual clutter.

loss of generality, we will assume v must intersect all grounded triangles in S except for those labelled x_i, x_{i+1}, y_j with $i \neq j$ and $i+1 \neq j$. We will first show that v intersects all other original vertices by the following:

Lemma 6.1.2. *Let S be an instance of* Outer Triangle Clique *transformed from an instance of* Pair-Orient Max Clique. *Then each grounded triangle in S which represents an original vertex intersects the origin o.*

Proof. Recall the polygon P during construction which forms the $4K$ thin grounded triangles, each of which are placed overlapping a side of P. For this proof, we will consider the sides of P and the thin grounded triangles in S as referring to the same line object. Let C be the inscribed circle of P. Therefore, P and C have the same center, which is o. As P is a regular polygon, each line lies tangent to C. Therefore, for any two lines $x \in X$ and $y \in Y$, their bisector will pass through o.

Consider now the construction of a dual ray $I_{y_j}^{x_{i+1}}$. Let a be the angle between x_i and the bisector of x_i and y_j. As well, b is to be defined as the internal angle of $I_{y_j}^{x_{i+1}}$ (Fig. 6.6 (Left)). If we define W as the intersection point of x_i y_j, then clearly, $\angle x_i, W, y_j = 2a$. Since $I_{y_j}^{x_{i+1}}$ lies on the bisector of x_i and y_j, if it can be shown that $b \geq 2a$, then it must be that the grounded triangle formed by $I_{y_j}^{x_{i+1}}$ intersects o.

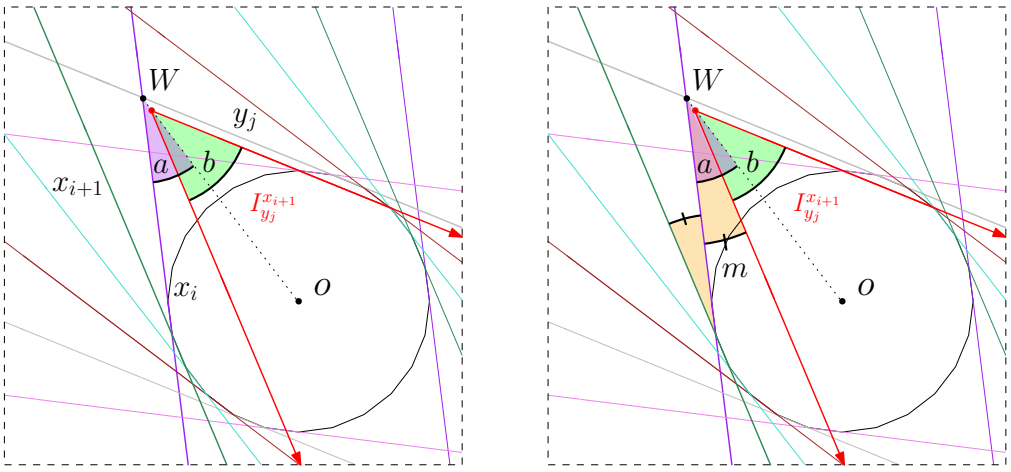

Figure 6.6: (Left) Visualization of the angles a and b. (Right) Visualization of the angle m.

This can be shown as follows. Let m be the minimum angle formed between two intersecting lines. Though geometry, we can calculate that $m = \frac{360}{4k}$. That is, m is the both the exterior angle of P, and the minimum angle for any two lines of P (Fig. 6.6 (Right)). Therefore, since x_i and x_{i+1} are consecutive lines, it must be that $2a - b = m$. We have that $a \geq m$, and so $a \geq 2a - b$, which simplifies to $b \geq 2a$.

Therefore, all dual rays, when transformed into grounded triangles, intersect the point o. □

By this lemma, it must be that v intersects all original vertices. Then it remains to show that v intersects all subdivision vertices except for x_i, x_{i+1}, and y_j.

Consider the placement of a dual ray $I_{y_j}^{x_{i+1}}$. Without loss of generality, assume $I_{y_j}^{x_{i+1}}$, I_1, and I_2 are in quadrant 2, and that I_1 is closer to o then I_2. Recall that the intersection points used in its formation I_1 and I_2 formed by $x_i \cap y_j$, and $x_{i+1} \cap y_j$ respectively. By its definition, we have that $I_{y_j}^{x_{i+1}}$ is merely a translation of the same dual ray positioned at I_2. Since I_2 and I_1 both lie on y_j, and I lies on the bisector of x_i and y_j, we have that $I_{y_j}^{x_{i+1}}$ cannot intersect x_{i+1} and y_j. Furthermore, we can conclude $I_{y_j}^{x_{i+1}}$ does not intersect x_i by its construction. It remains to show that each other thin grounded triangles must intersect v.

Assume for contradiction that there exists an additional line l which does not intersect $I_{y_j}^{x_{i+1}}$. The position of l can be place in one of three ways: l is not tangent to C, l is tangent to C and appears between x_i and x_{i+1}, or l is tangent to C and passes between I_1 and $I_{y_j}^{x_{i+1}}$. Case 1 yields a contradiction as such a line is not allowed in construction. Case 2 yields a construction as x_i and x_{i+1} would not longer be consecutive. Case 3 yields a contradiction as for any choice of l, we can move $I_{y_j}^{x_{i+1}}$ along the bisector $I_1 o$ arbitrarily close to I_1 so that l intersects our dual ray. In this way **P1** is proven.

Preserving P2: Clearly, each division vertex represented as a line will be adjacent to all other lines, except for its parallel, which by construction will have the same subscript. Now we need only show each thin grounded triangle intersects exactly all but one dual ray. By the algorithm, each dual ray is constructed from three unique lines. That is, no subdivision vertex is used twice, or excluded entirely, when creating an instance of S since the choices of vertices are defined directly from G. Therefore, as **P1** was proven, it must be that each line is used in exactly one dual ray. □

6.2 Summary

Similar to Section 5, the proof of showing Outer Triangle Clique naturally extends itself to similar constructions by making some small modifications to the end results. In this case, we may choose not to convert the dual rays and lines from constructing S into grounded triangles. Alternatively, we can choose to keep the division vertices (lines) as they are and transform the dual rays into grounded triangles. Lastly, we may also treat the lines as half-planes, as all intersections can be properly maintained by having each half-plane not intersect the side of its associated dual ray. With these multiple variations, it stands that many adjacent maximum clique problems are also NP-hard.

For Outer Triangle Clique, it would not be sufficient to prove this using standard co-2-subdivision. This is because as in Section 5, imposing an ordering to the subdivided vertices means we can exploit certain geometric facts. In this case, the consecutive nature of two lines x_i and x_{i+1} is what allows us to prove property **P1** in Lemma 6.1.2. Due to the success of pair-oriented labelling on the outer triangle graph, it gives credit to the potential of this technique being used elsewhere.

7 Conclusion

In this section we wrap up the findings of this thesis by reviewing the major contributions. The research questions are reviewed, and additional time is taken to discuss various open problems which arise from this work.

7.1 Results

This thesis has presented a focus on geometric intersection graphs and the ease or difficulty in calculating the maximum clique for various classes. The exploration into the hardness of maximum clique for existing classes of intersection graphs laid a foundation seen in the literature review. The co-2-subdivision technique traditionally used to show such problems are NP-hard was investigated to a depth of understanding why it is so useful. This led to a collection of three primary results all centered around maximum clique in intersection graphs.

In Section 4, a $O(n^{2.5} \log n)$ time algorithm was provided to improve the existing algorithm for unit disk graphs. This represents the ever moving improvement on difficult classes of intersection graphs. The implementation of a more recursive algorithm may also pave the way for potentially more sophisticated divide-and-conquer algorithms in the field. In the context of research question **RQ1**, we responded to it directly by successfully finding an algorithm which calculates the maximum clique for a unit disk graph faster than $O(n^3 \log n)$.

In Section 5 we introduced a structure and its associated technique, known as pair-oriented labelling, which acts as the levering method discussed in research question **RQ2**. This concept allows the process of co-2-subdivision to be applied for previously difficult problems, as shown in Disk-Axis Clique. Section 6 expands on this by showing the hardness for a novel problem in the outer triangle graph. In this way, we displayed how pair-oriented labelling can be leveraged to prove the NP-hardness of finding maximum clique in other intersection graph classes.

It is worth noting that Disk-Axis Clique is known to be APX-hard [6] following their reduction from MIPA. Some of the other intersection graph classes for which the maximum clique problem has been proved to be APX-hard using the co-2k-subdivision approach are intersection graphs of ellipses [3], triangles [3], string graphs [36], grounded string graphs [32], and so on. Following this, similar reductions could be made to show that Outer Triangle Clique and Disk-Axis Clique are APX-hard from the usage of pair-oriented labelling.

7.2 Future Work and Open Questions

The improvement of maximum clique on unit disk graphs is a strong result in its own right by increasing the efficiency for a well-known algorithm. As is with all results in disk graphs, any progress is good progress. Evidently continuing work on the main open question posed by Ambühl and Wagner [3] for the hardness of a general disk graph is important work worth pursuing. There exist many adjacent problems however. One of which is a disk graph with two sizes of disks. Generally speaking, we may consider disks of either size 1 unit, or size 2 unit. At the time of writing this, no such solution is known despite the seemingly trivial addition.

Open Question 1: Let G be a disk graph where each disk has radius 1 unit or radius 2 unit. Does there exist a polynomial time algorithm to calculate the maximum clique of G?

The pair-oriented labelling introduced in Section 5 holds the minor caveat that the input cubic graph must be a Hamiltonian cubic graph. This distinction bears no extra cost to its use as a way of showing certain intersections graphs are NP-hard, but it naturally begs the question of whether pair-oriented labelling can be applied to non-Hamiltonian cubic graphs. Resolving this question may lead to a greater understanding of the interaction between labelling and orienting edges. As it stands, pair-oriented labelling requires a Hamiltonian cubic graph in its construction by means of ensuring each vertex has at least one incoming and one outgoing edge. Strictly speaking, a vertex can have all incoming or outgoing edges and still be satisfied, and so it may not be necessary to include the Hamiltonian cycle as a part of constructing a pair-oriented labelling.

Open Question 2: Let $P(G)$ be the pair-oriented labelling for G. Does $P(G)$ exist for every cubic graph G?

Section 6 stands as primarily an example to the use of pair-oriented labelling on new problems. The geometry present in the proof for showing Outer Triangle Clique is NP-hard holds some hope in improving previous work, or adapting this result to prove new ones entirely. Specifically, the proof that ray (and segment) intersection graphs are NP-hard for maximum clique [12] is a fairly dense proof. Given the nature of using dual rays, and the flexibility in representing division vertices as lines, there is some speculation that this proof may be adaptable to provide a more streamlined result for ray intersection graphs.

Open Question 3: Can the maximum clique problem be shown to be NP-hard for ray intersection graphs using the co-2-subdivision approach?

More generally speaking, the constructive proof in sections 5 and 6 for Disk-Axis Clique and Outer Triangle Clique follow a similar outline. By arranging lines in a conveniently permuted way, and then intelligently introducing a specific shape (axis-aligned rectangle or grounded triangle), one can show these intersection graphs to be hard. It stands to reason that this structure can be extended further to both existing or new intersection graphs. A similar structure had been seen in [6]. However, with the introduction of pair-oriented labelling, more flexibility may be available in construction than was previously thought. As a result, applying this kind of constructive structure in other hardness proofs may provide useful results, and it is one worth studying further. Perhaps some overlap may be discovered between this and disk graphs.

References

[1] Meriem Adraoui, Asmaâ Retbi, Mohammed Khalidi Idrissi, and Samir Bennani. Maximal cliques based method for detecting and evaluating learning communities in social networks. *Future Gener. Comput. Syst.*, 126:1–14, 2022.

[2] Alok Aggarwal, Hiroshi Imai, Naoki Katoh, and Subhash Suri. Finding k points with minimum diameter and related problems. *J. Algorithms*, 12(1):38–56, 1991.

[3] Christoph Ambühl and Uli Wagner. The clique problem in intersection graphs of ellipses and triangles. *Theory Comput. Syst.*, 38(3):279–292, 2005.

[4] Marthe Bonamy, Édouard Bonnet, Nicolas Bousquet, Pierre Charbit, Panos Giannopoulos, Eun Jung Kim, Pawel Rzazewski, Florian Sikora, and Stéphan Thomassé. EPTAS and subexponential algorithm for maximum clique on disk and unit ball graphs. *J. ACM*, 68(2):9:1–9:38, 2021.

[5] Édouard Bonnet, Panos Giannopoulos, Eun Jung Kim, Pawel Rzazewski, and Florian Sikora. QPTAS and subexponential algorithm for maximum clique on disk graphs. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry, SoCG*, volume 99 of *LIPIcs*, pages 12:1–12:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[6] Édouard Bonnet, Nicolas Grelier, and Tillmann Miltzow. Maximum clique in disk-like intersection graphs. *CoRR*, abs/2003.02583, 2020.

[7] Prosenjit Bose, Paz Carmi, J. Mark Keil, Anil Maheshwari, Saeed Mehrabi, Debajyoti Mondal, and Michiel Smid. Computing maximum independent set on outerstring graphs and their relatives. *Comput. Geom.*, 103:101852, 2022.

[8] Prosenjit Bose, Saeed Mehrabi, and Debajyoti Mondal. Parameterized complexity of two-interval pattern problem. *Theor. Comput. Sci.*, 902:21–28, 2022.

[9] Heinz Breu. *Algorithmic aspects of constrained unit disk graphs*. PhD thesis, 1996.

[10] Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is np-hard. *Comput. Geom.*, 9(1-2):3–24, 1998.

[11] Sergio Cabello. Maximum clique for disks of two sizes. Open problems from Geometric Intersection Graphs: Problems and Directions, CG Week Workshop, 2015.

[12] Sergio Cabello, Jean Cardinal, and Stefan Langerman. The clique problem in ray intersection graphs. *Discret. Comput. Geom.*, 50(3):771–783, 2013.

[13] Miroslav Chlebík and Janka Chlebíková. The complexity of combinatorial optimization problems on d-dimensional boxes. *SIAM J. Discret. Math.*, 21(1):158–169, 2007.

[14] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discret. Math.*, 86(1-3):165–177, 1990.

[15] Mark Theodoor De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational geometry: algorithms and applications*. Springer Science & Business Media, 2000.

[16] David Eppstein. Graph-theoretic solutions to computational geometry problems. *CoRR*, abs/0908.3916, 2009.

[17] Jared Espenant, J. Mark Keil, and Debajyoti Mondal. Finding a maximum clique in a disk graph. In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium on Computational Geometry, SoCG 2023, June 12-15, 2023, Dallas, Texas, USA*, volume 258 of *LIPIcs*, pages 30:1–30:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[18] Jared Espenant, J. Mark Keil, and Debajyoti Mondal. Finding a maximum clique in a disk graph. *CoRR*, abs/2303.07645, 2023. To Appear in SoCG 2023.

[19] David Eu, Eric Guévremont, and Godfried T. Toussaint. On envelopes of arrangements of lines. *J. Algorithms*, 21(1):111–148, 1996.

[20] Stefan Felsner, Rudolf Müller, and Lorenz Wernisch. Trapezoid graphs and generalizations, geometry and algorithms. *Discret. Appl. Math.*, 74(1):13–32, 1997.

[21] Aleksei V. Fishkin. Disk graphs: A short survey. In Klaus Jansen and Roberto Solis-Oba, editors, *Approximation and Online Algorithms, First International Workshop, WAOA*, volume 2909 of *Lecture Notes in Computer Science*, pages 260–264. Springer, 2003.

[22] Herbert Fleischner, Gert Sabidussi, and Vladimir I. Sarvanov. Maximum independent sets in 3- and 4-regular hamiltonian graphs. *Discret. Math.*, 310(20):2742–2749, 2010.

[23] Mathew C. Francis, Daniel Gonçalves, and Pascal Ochem. The maximum clique problem in multiple interval graphs. *Algorithmica*, 71(4):812–836, 2015.

[24] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[25] Udaiprakash I. Gupta, D. T. Lee, and Joseph Y.-T. Leung. Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12(4):459–467, 1982.

[26] John Hershberger and Subhash Suri. Finding tailored partitions. *J. Algorithms*, 12(3):431–463, 1991.

[27] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[28] Mark L. Huson and Arunabha Sen. Broadcast scheduling algorithms for radio networks. In *Proceedings of MILCOM'95*, volume 2, pages 647–651. IEEE, 1995.

[29] Hiroshi Imai and Takao Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *J. Algorithms*, 4(4):310–323, 1983.

[30] Hiroshi Imai and Takao Asano. Efficient algorithms for geometric graph search problems. *SIAM J. Comput.*, 15(2):478–494, 1986.

[31] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[32] J. Mark Keil, Debajyoti Mondal, Ehsan Moradi, and Yakov Nekrich. Finding a maximum clique in a grounded 1-bend string graph. *Journal of Graph Algorithms and Applications*, 26(4), 2022.

[33] Jan Kratochvíl and Jaroslav Nešetřil. Independent set and clique problems in intersection-defined classes of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 31(1):85–93, 1990.

[34] George B. Mertzios. The recognition of triangle graphs. *Theor. Comput. Sci.*, 438:34–47, 2012.

[35] George B. Mertzios. The recognition of simple-triangle graphs and of linear-interval orders is polynomial. *SIAM J. Discret. Math.*, 29(3):1150–1185, 2015.

[36] Matthias Middendorf and Frank Pfeiffer. The max clique problem in classes of string-graphs. *Discret. Math.*, 108(1-3):365–372, 1992.

[37] Michael Ian Shamos and Dan Hoey. Closest-point problems. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 151–162, 1975.

[38] Xiaohong Shi, Luo Liang, Yan Wan, and Jin Xu. A finding maximal clique algorithm for predicting loop of protein structure. *Appl. Math. Comput.*, 180(2):676–682, 2006.

[39] Qinghua Wu and Jin-Kao Hao. A review on algorithms for maximum clique problems. *Eur. J. Oper. Res.*, 242(3):693–709, 2015.