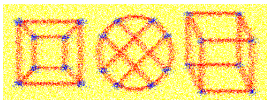
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 26, no. 4, pp. 553–575 (2022)
DOI: 10.7155/jgaa.00608

Finding a Maximum Clique in a Grounded 1-Bend String Graph

J. Mark Keil[1] *Debajyoti Mondal*[1] *Ehsan Moradi*[1] *Yakov Nekrich*[2]

[1]Department of Computer Science,
University of Saskatchewan, Saskatoon, Canada
[2]Department of Computer Science,
Michigan Technological University, Michigan, USA

Submitted:	June 2021	Reviewed:	March 2022
Revised:	May 2022	Reviewed:	October 2022
Revised:	November 2022	Accepted:	November 2022
Final:	December 2022	Published:	December 2022
Article type:	Regular paper	Communicated by:	Ignaz Rutter

Abstract. A grounded 1-bend string graph is an intersection graph of a set of polygonal lines, each with one bend, such that the lines lie above a common horizontal line ℓ and have exactly one end point on ℓ. We show that the problem of finding a maximum clique in a grounded 1-bend string graph is APX-hard, even for strictly y-monotone strings. For general 1-bend strings, the problem remains APX-hard even if we restrict the position of the bends and end points to lie on at most three parallel horizontal lines. We give fast algorithms to compute a maximum clique for different subclasses of grounded segment graphs, which are formed by restricting the strings to various forms of L-shapes.

1 Introduction

A *geometric intersection graph* consists of a set of geometric objects (e.g., disks, rectangles, etc.) representing the nodes of the graph, where two nodes are adjacent if and only if the corresponding objects intersect. Intersection graphs are intriguing from the perspective of algorithm design as

A preliminary version of the paper was presented at the 32nd Canadian Conference on Computational Geometry (CCCG 2020) [14].

E-mail addresses: mark.keil@cs.usask.ca (J. Mark Keil) dmondal@cs.usask.ca (Debajyoti Mondal) e.moradi@usask.ca (Ehsan Moradi) yakov@mtu.edu (Yakov Nekrich)

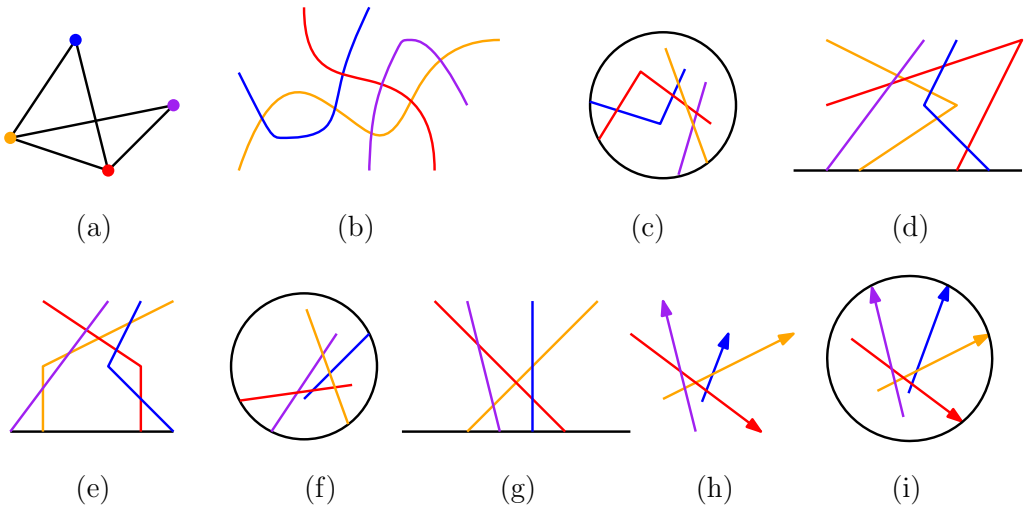

Figure 1: (a) A graph G. (b) A string representation. (c) A 1-bend outerstring representation. (d) A grounded 1-bend string representation. (e) An outersegment representation. (f) A grounded segment representation. (g) A ray intersection graph. (h) Transformation from a ray intersection representation to an outersegment representation.

many optimization problems that are NP-hard for general graphs, are known to be polynomial-time solvable for different classes of intersection graphs [13, 17, 19, 24]. Intersection graphs appear naturally in many applied scenarios [5, 10, 17], e.g., wireless communication networks are commonly modeled with disk intersection graphs, scheduling problems are often modeled on intersection graphs of intervals in one dimension, and so on. Intersection graphs in $\mathbb{R}^2$ have been extensively studied in graph drawing area to intuitively visualize the relational information present in a graph [23]. We refer the readers to [17] for more details on intersection graphs.

String Graphs: In this paper we restrict our attention to *strings*, which are simple curves in $\mathbb{R}^2$, and to objects obtained by imposing various geometric restrictions to them. A *string graph* is a geometric intersection graph, where each node represents a string and each edge represents an intersection between the two strings that correspond to the end points of the edge. Figure 1(a) illustrates a string graph and Figure 1(b) illustrates its string intersection representation. In 1959, Benzer [2] first introduced string graphs while exploring the topology of genetic structures. Later, Sinden [22] explored string graphs in the context of electrical networks. A rich body of literature has examined string graphs since then to understand the structure of string graphs [6, 16, 21].

A string is often modeled with a polygonal chain and sometimes, with an orthogonal polygonal chain. In this paper, unless otherwise stated, we consider a string to be a polygonal chain where its segments may have arbitrary slopes. A *k-bend string* is a polygonal chain with at most k bends or $(k+1)$ segments.

A number of restrictions on the strings have been examined in the literature [3, 12]. Outerstring graphs and grounded string graphs are two widely studied classes of graphs that resulted from such restrictions. An *outerstring graph* is a string graph, where each string must lie inside a disk, with exactly one end point on the boundary of the disk. If each string in an outerstring representation contains at most k bends, then we call the corresponding graph a *k-bend outerstring*

graph. Figure 1(c) illustrates a 1-bend outerstring representation. A *grounded string graph* is a string graph, where each string must lie above a common horizontal line ℓ, with exactly one end point on ℓ. The line ℓ is referred to as a *ground line*. If each string in a grounded segment representation contains at most k bends, then we call the corresponding graph a *k-bend grounded segment graph*. Figure 1(d) illustrates a grounded 1-bend string representation. A grounded segment graph is called *strictly y-monotone* if admits an intersection representation with strings that are strictly monotone with respect to the y-axis (Figure 1(e)).

Although the outerstring graphs and grounded string graphs are the same for general strings, they can be different when we put restrictions on the number of bends that a string can have. For example, if we restrict the strings to be straight line segments, the resulting outerstring (respectively, grounded segment) graph is called an *outersegment* (respectively, *grounded segment*) graph. Figures 1(f)–(g) illustrate such graphs. The grounded segment graph class is known to be a proper subclass of the outersegment graph class [6]. We refer the readers to [12] for more details on various subclasses of string graphs, and their inclusion or separation properties.

Maximum Independent Set and Maximum Clique Problems: A *maximum independent set* in a graph G is a largest set of vertices S in G such that no two vertices in S are adjacent in G. A *maximum clique* in G is a largest set of vertices that are pairwise adjacent in G. The hardness of independent set and maximum clique problems in general graphs inspired researchers to examine these problems for restricted intersection graph classes [11, 15, 19, 24]. Both maximum independent set and maximum clique problems are known to be NP-complete in general segment intersection graphs, i.e., the intersection graph of a set of segments in $\mathbb{R}^2$ [15]. Since a segment is a 0-bend string, this implies that these problems are NP-complete for string graphs. Fox and Pach [11] gave subexponential time algorithms for computing a maximum independent set for string graphs. They also considered *k-intersecting string graphs*, which are string graphs with the additional property that one string cannot intersect another string more than k times. They gave algorithms for approximating independent set and maximum clique for k-intersecting string graphs.

While the maximum independent set problem is NP-complete for string graphs, Keil et al. [13] showed the problem to be polynomial-time solvable for outerstring graphs with a given outerstring representation of polynomial size [13]. Given an outerstring representation, where each segment is represented as a polygonal chain, they showed how to compute a maximum independent set in $O(s^3)$ time, where s is the total number of segments in the representation. Bose et al. [3] gave an $O(n^2)$-time algorithm when the strings are y-monotone polygonal paths of constant length with segments at integral coordinates. They also showed this to be the best possible under strong exponential time hypothesis.

Middendorf and Pfeiffer [18] showed the maximum clique problem to be NP-hard for axis-aligned 1-bend strings, even when the strings are of two types: ⅂ and L. They also showed the problem to be polynomial-time solvable for two cases: (a) For the strings of type ⅂ and ⅂, and (b) for grounded segments when the free end points of the segments lie on a fixed number of horizontal lines.

A *ray intersection graph* is a geometric intersection graph, where each node represents a ray and each edge represents an intersection between the two rays that correspond to the end points of the edge. Figure 1(h) illustrates a ray intersection graph. Cabello et al. [4] proved the maximum clique problem to be NP-complete even for ray intersection graphs. Since one can enclose a ray intersection graph inside a circle such that all the rays hit the perimeter (e.g., Figure 1(i)), their result implies NP-completeness for computing a maximum clique in an outersegment graph. Therefore,

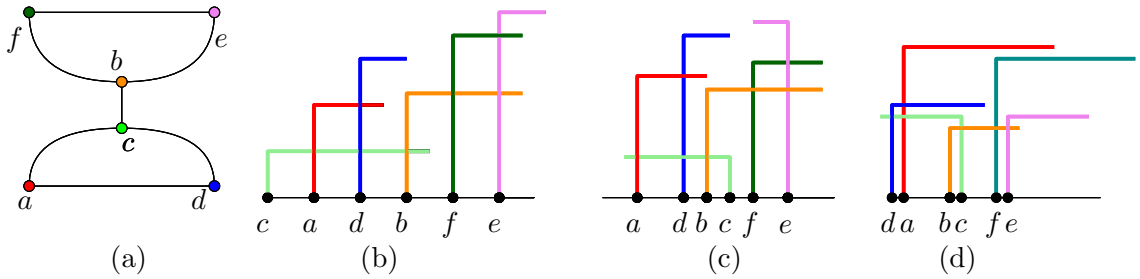

Figure 2: (a) A graph G. (b) An 1-sided L-shape representation. (c) A square L-shape representation. (d) A two-sided L-shape representation.

an interesting question that arises in this context is whether the maximum clique problem remains NP-complete for grounded segment graphs. While the problem remains open, in this paper, we show APX-hardness for two subclasses of grounded 1-bend string graphs.

A rich body of research examines the recognizability of various classes of string graphs [6,16,21]. The recognition problem is known to be NP-hard even for the intersection graphs of L-shapes [9]. Throughout this paper, whenever we examine an intersection model, we assume that the input graph comes with a representation satisfying that intersection model. For the grounded string graphs, we assume the x-axis to be the ground line.

1.1 Contributions

We first prove that the problem of computing a maximum clique in a grounded 1-bend string graph is APX-hard, even for strictly y-monotone strings. Although grounded 1-bend strings include grounded segments, our reduction heavily relies on bends and thus the maximum clique problem remains open for grounded segment graphs.

We then show that the problem remains NP-hard for grounded 1-bend strings even when the bend and end points of the strings (not necessarily y-monotone) are restricted to lie on three horizontal lines. This is interesting since the maximum clique problem is polynomial-time solvable for grounded segment graphs with the end points on a set of fixed horizontal lines [18].

We give fast polynomial-time algorithms for some axis-aligned grounded 1-bend string graphs. In particular, here a grounded 1-bend string is an inverted L-shape or the vertical mirror reflection of an inverted L-shape. For simplicity we will refer to them as L-shapes with exactly one end point on the ground line (e.g., Figures 2(b)–(d)). The reason to investigate the maximum clique problem on grounded L-shape graphs is that they belong to the class of grounded segment graphs for which the problem is open.

The time-complexity of the maximum independent set problem has been examined for various subclasses of grounded L-shape graphs [3]. It is thus natural to examine the maximum clique problem on them. Here we consider three types of grounded L-shape intersection representations and throughout the paper we assume the L-shapes are in general position, i.e., no two segments in the intersection representation lie on the same horizontal or vertical line. An L-shape intersection representation is *1-sided* (Figure 2(a)–(b)), if the L-shapes in the representation all turn clockwise, or all turn anticlockwise. Otherwise, the representation is *two-sided* (Figure 2(c)). In a *square L-shape representation*, the horizontal and vertical segments of every L-shape are of the same length (Figure 2(d)). These graph classes are already known in the literature [3, 12]. It is known that

there exist grounded 2-sided L-shape graphs that are not grounded 1-sided L-shape graphs [12]. In this paper, we show that there exist grounded 2-sided L-shape graphs that are not grounded 2-sided square L-graphs, and there exist grounded 2-sided square-L graphs that are not grounded 1-sided L-shape graphs. Therefore, it is worth investigating the time-complexity of the maximum clique problem on each of the above graph classes independently.

The results are summarized in the following table. Note that the class of 2-sided grounded L-shapes is known to be a proper subclass of grounded segment graph class [12]. Therefore, our results do not settle the time complexity question for computing a maximum clique in a grounded segment graph.

Graph Class	Complexity	Reference
Grounded 1-bend string graphs	APX-hard	Section 2
Grounded 2-sided L-shape graphs	$O\left(\frac{n^2 \log^2 n}{(\log \log n)^2}\right)$	Section 3
Grounded 2-sided square-L graphs	$O(n^2 \log \log n)$	Section 4
Grounded 1-sided L-shape graphs	$O(n^2 \log \log n)$	Section 5

2 APX-hardness

In this section we first show that finding a maximum clique in a grounded 1-bend string graph is an APX-hard problem, even when each string is strictly y-monotone (Theorem 1). We then prove that the maximum clique problem for grounded 1-bend string graphs remains APX-hard, even when the bends and end points are restricted to lie on three horizontal lines (Theorem 2).

Theorem 1 *The maximum clique problem is APX-hard for grounded 1-bend string graphs, even when the strings are strictly y-monotone.*

Theorem 2 *The maximum clique problem is APX-hard for grounded 1-bend string graphs, even when the bends and end points are restricted to lie on three horizontal lines above the ground line.*

We now introduce a few definitions and notation, which will be used to present the NP-hardness results in the subsequent sections (Sections 2.1–2.2).

A 2-*subdivision* of a graph is obtained by replacing each edge (u, v) of G with a path (u, d_1, d_2, v) of 2 division vertices. Let G be a graph. The *complement of* G is a graph on the same set of vertices as that of G and there exists an edge in the complement if and only if the edge does not exist in G. We denote the complement of G as $\overline{G}$. A *cubic graph* is a graph where every vertex is of degree three. Thus a cubic graph with n vertices has $3n/2$ edges.

Given a 1-bend string, we will refer to its two end points as *fixed* or *free* depending on whether they lie on the ground line or not. Similarly, we call its segments as *fixed* or *free* depending on whether the segment is adjacent to the ground line or not.

2.1 APX-Hardness results for Grounded 1-Bend String Graphs with y-Monotone Strings

In this section we prove Theorem 1. We first prove that the complement of a 2-subdivision of a cubic graph admits a strictly y-monotone grounded 1-bend string representation (Lemma 1). We then show that finding a maximum clique is APX-hard for such string graphs.

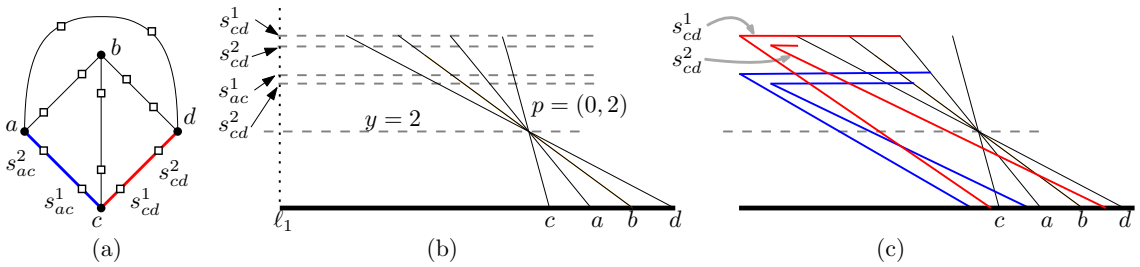

Figure 3: Illustration for the (a) graph G, and (b) the construction of the original strings. (c) Construction of a grounded 1-bend string representation for $\overline{G}$.

Lemma 1 *Let H be a cubic graph and let G be a 2-subdivision of H. Then the complement $\overline{G}$ of G admits a strictly y-monotone grounded 1-bend string representation and such a representation can be constructed in polynomial time.*

Proof: While constructing the y-monotone grounded 1-bend string representation for $\overline{G}$, we will refer to the strings corresponding to the vertices that originally belong to H as the *original strings*, and the other strings as the *division strings*. The vertices of $\overline{G}$ that originally belong to H are called the *original vertices* and the remaining vertices are *division vertices*.

Note that the original vertices form a clique in $\overline{G}$. Therefore, they must mutually intersect in the string representation. We now describe the construction for the original strings. We label the vertices of H from 1 to n. For each vertex $i = 1 \ldots, n$ in H, we construct an original string, which is a straight line segment with end points $(i, 0)$ and $(-3ni, 6n + 2)$. All these lines pass through the point $p = (0, 2)$. Consequently, we obtain a set of pairwise intersecting strings. Figure 3(a) illustrates an example of a 2-subdivision of a cubic graph G. Figure 3(b) illustrates the construction of the original strings for $\overline{G}$.

We now describe the construction for the division strings. For each edge of H, G contains a pair of division vertices. These two division vertices must not be adjacent in $\overline{G}$, but they must be adjacent to all the other division vertices in $\overline{G}$. Moreover, each division vertex in $\overline{G}$ must be adjacent to all the original vertices except for the one original vertex that is adjacent to it in G. We construct the division strings such that they satisfy these properties, e.g., the blue and red strings in Figure 3(c). The idea is to create pairs of non-intersecting strings such that these pairs mutually intersect and the two strings in each pair intersect the necessary original strings. The detailed construction is as follows.

Let ℓ_1 be the vertical segment that starts at $(-3n^2 - 1, 6n + 2)$ and hits the ground line. Figure 3(b) illustrates ℓ_1 with a dotted line for the construction being carried out for the example graph of Figure 3(a). There are at least $6n$ horizontal lines through integral coordinates between $y = 3$ and $y = 6n + 6$. This number is larger than $3n$, i.e., the number of division vertices in $\overline{G}$. We use these lines to create the division strings.

We number the edges of H from 1 to $(3n/2)$. Let (c, d) be the jth edge in H and let s^1_{cd}, s^2_{cd} be the corresponding division vertices in G. The vertex s^1_{cd} (resp., s^2_{cd}) is adjacent to all the original vertices and division vertices of $\overline{G}$, except for c (resp., d) and s^2_{cd} (resp., s^1_{cd}). We now construct a pair of division strings that realize these adjacencies.

Let c and d be the tth and rth vertex of H. Assume without loss of generality that the string of c appears to the left of the string of d on the ground line. The division string for s^1_{cd} starts

at the ground line, makes a bend at ℓ_1, and then intersects all the original strings that lie to the left of the string for c. In particular, the string starts at $(t - \frac{1}{j}, 0)$, makes a right turn at $q = (-3n^2 - 1, 6n - j + 3)$ following the line $y = (6n - j + 3)$, and stops as soon as it intersects all the strings that appear before the string of c on this line. The division string for s^2_{cd} starts at $w = (r - \frac{1}{j}, 0)$, makes a right turn at the intersection point of the lines qw and $y = (6n - j + 2)$, and continues until it intersects all the strings that appear before the string of d. Figure 3(c) illustrates the construction of the division strings for the complement of the example graph in Figure 3(a), where (c, d) is labelled as the first edge, (a, c) is labelled as the second edge, and so on.

We now verify whether the division strings satisfy the necessary adjacencies in $\overline{G}$. We again consider the jth edge (c, d) of H. We first examine whether its corresponding division vertices in $\overline{G}$ are adjacent to the appropriate set of original vertices in $\overline{G}$. Since the permutation of the original strings on the ground line is opposite to the permutation on the line $y = 6n + 2$, it is straightforward to verify that the string for s^1_{cd} intersects all the original strings except the one for c. Similarly, the string for s^2_{cd} intersects all the original strings except the one for d.

We now examine whether s^1_{cd} and s^2_{cd} are adjacent to the appropriate set of division vertices in $\overline{G}$. By construction, the strings of s^1_{cd} and s^2_{cd} are disjoint. Therefore, it suffices to check whether they intersect all the previously added division strings. Let z be a division string which is added prior to the strings of s^1_{cd} and s^2_{cd}. By construction, the line determined by the fixed segment of z intersects the required original strings below the line $y = 2$ and keeps the bend points of s^1_{cd} and s^2_{cd} to its left half-plane. Since the free segments of s^1_{cd} and s^2_{cd} lie above $y = 2$ and reach the original strings, they both intersect the fixed segment of z. This completes the grounded string representation for $\overline{G}$.

Since the coordinates explicitly described above are polynomial in n, the intersection of the line segments required to carry out the construction also have coordinates of polynomial number of bits. Hence one can compute the string representation in polynomial time. Note that the strings that we computed are y-monotone. To make the strings strictly y-monotone, one can carry out the same construction for division strings with a set of slanted parallel lines of small positive slope for the free segments of the division strings, instead of horizontal ones. Note that all the original strings in our construction pass through a common point. However, one can perturb the free end points of the original strings to construct a representation where no three strings intersect at a common point. □

We are now ready to prove Theorem 1.

Proof of Theorem 1: We reduce the maximum independent set problem in 2-subdivisions of cubic graphs, which is known to be APX-hard [8]. Let G be a 2-subdivision of a cubic graph H. We compute the complement graph $\overline{G}$. By Lemma 1, $\overline{G}$ admits a grounded 1-bend string representation with strictly y-monotone string.

Note that an independent set in a graph corresponds to a clique in its complement, and vice versa. Therefore, the existence of a $(1 - \varepsilon)$-approximation algorithm for the maximum clique in $\overline{G}$ implies a $(1 - \varepsilon)$-approximation algorithm for the maximum independent set in H. □

2.2 APX-Hardness Results for Grounded 1-Bend String Graphs on a Few Lines

In this section we prove Theorem 2.

Proof of Theorem 2: The proof structure is the same as the proof of Theorem 1 except that we slightly modify the construction presented at Lemma 1 in the previous section such that the bends and end points of the strings lie on at most three lines above the ground line. In particular, we show that $\overline{G}$ admits a grounded 1-bend string representation where the bends and end points are restricted to lie on three horizontal lines above the ground line.

We define three horizontal lines ℓ_1, ℓ_2, ℓ_3 above the ground line with increasing heights, as illustrated in Figure 4. We omit the coordinate details for this construction. It is straightforward to use a very similar approach as in Lemma 1 to compute the representation with coordinates of polynomial number of bits.

We create the original strings such that bend-points and free end points lie on ℓ_2 and ℓ_1, respectively. We also ensure that the order of the free end points on ℓ_1 is opposite to the order of the fixed end points. Therefore, the original strings mutually intersect, which is consistent to the property that the original vertices of $\overline{G}$ are mutually adjacent in $\overline{G}$.

We now construct the division vertices. Consider the jth edge (c, d) of H. The division vertices s^1_{cd} and s^2_{cd} of $\overline{G}$, that correspond to the same edge in the cubic graph H must be non-adjacent in H. However, they are adjacent to all other division vertices. Furthermore, s^1_{cd} and s^2_{cd} are adjacent to all the original vertices except c and d, respectively.

Let q be a point to the right of all the free end points of the original strings on ℓ_1. A pair of division strings start at the ground line near their corresponding original strings (as in our earlier construction), but their bends are placed on ℓ_2 and ℓ_3 such that the strings remain disjoint. We also ensure that fixed segments of these strings lie to the right of q. Consider now a division string z that starts at the ground line and reaches ℓ_2 or ℓ_3. If only a subset of the required intersections is realized at its fixed segment, then the free segment is created by connecting the bend-point to an appropriate point q' to the left of q on ℓ_1. If all the required intersections are realized at its fixed segment, then we create the free segment by connecting q and its bend-point. Since the permutation of the fixed end points of the original strings is the reverse of the permutation of their free end points, such a point q' must exist.

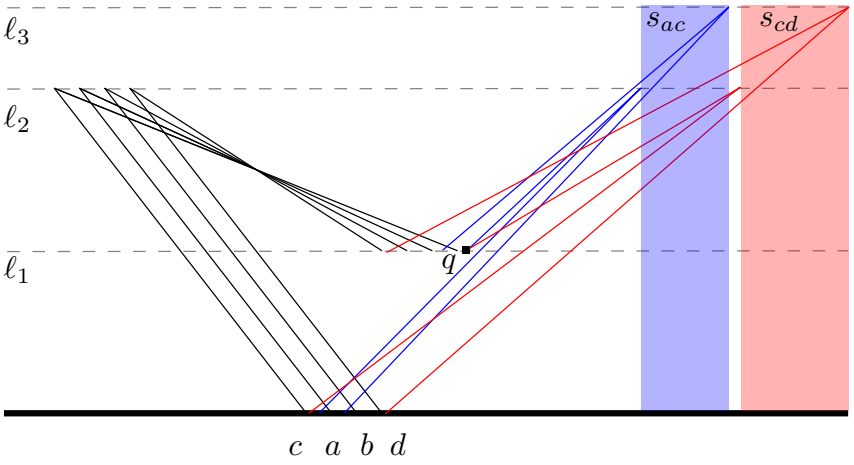

Figure 4: Construction of a grounded string representation for $\overline{G}$ on three lines.

The division vertices are created in pairs and their bend-points are placed to the right of all the previously created bend-points, as illustrated using the vertical stripes in Figure 4. Note that every

newly created string z needs to reach either q or to a point to the left of q on ℓ_1. Therefore, we can choose the new bend-point of z sufficiently far apart such that its free segment crosses all the previously added division strings at their fixed segments. This completes the required construction for $\overline{G}$.

The proof now follows from the observation that the existence of an $(1-\varepsilon)$-approximation algorithm $\mathcal{A}$ for the maximum clique problem in $\overline{G}$ implies an $(1-\varepsilon)$-approximation algorithm for the maximum independent set problem for G. □

3 Two-sided L-shapes

In this section we consider the case when the grounded strings are two-sided L-shapes. For simplicity, we first describe a dynamic program to compute a maximum clique on this class of graphs (Section 3.1), and give an $O(n^3)$-time algorithm to compute a maximum clique. We then improve the running time to $O((\frac{n \log n}{\log \log n})^2)$ in (Section 3.2).

3.1 Computing a Clique in $O(n^3)$ time

Let G be an intersection graph of two-sided L-shapes, and let Q be a maximum clique with at least two vertices in G. Let a and b be the highest and second-highest L-shape in Q, respectively, and without loss of generality assume that a appears to the left of b on the ground line (Figure 5). Then any other L-shape c in Q must be below the line ℓ determined by the horizontal segment of b. Furthermore, since c intersects both a and b, its end point on the ground line must be to the left of a or to the right of b. In other words, the interval $[a, b]$ on the ground line acts as a forbidden interval for the other vertices in the clique.

If c is the next highest L-shape after b in Q, then depending on whether it lies to the left or right of the interval $[a, b]$, the forbidden region for the remaining vertices in Q grows to either $[c, b]$ or $[a, c]$. Without loss of generality assume that c lies to the right of b, and the new forbidden region is $[a, c]$. Then an L-shape intersecting c and a must also intersect b, where b already belongs to Q. One can thus continue adding a new L-shape that intersect the L-shapes representing the current forbidden interval, without worrying about the L-shapes which have been chosen already. We use this idea to design the dynamic programming algorithm.

If G does not contain any edge, then the maximum clique size is 1. Otherwise, let $D(a, b)$ denote a maximum clique, where a, b, or b, a are the first and second highest L-shapes, and a lies to the left of b. Let c be an L-shape that intersects the vertical segments of a and b, and for an L-shape w, let w_x be its x-coordinate on the ground line. Then $D(a, b)$ is defined as follows.

$$D(a,b) = \begin{cases} 2, \text{ if } c \text{ doesn't exist}, \\ \max \begin{cases} \max\limits_{c_x < a_x} \{D(c,b)\} + 1 \\ \max\limits_{c_x > b_x} \{D(a,c)\} + 1, \quad \text{otherwise.} \end{cases} \end{cases}$$

We take the maximum over all pairs of L-shapes a, b in the input. We can use a 2-dimensional table $T(a, b)$ to store the solution of $D(a, b)$. The size of the dynamic programming table is $O(n^2)$, where n is the number of L-shapes. Computing each entry of the table requires $O(n)$ table look-up. Therefore, it is straightforward to compute the maximum clique in $O(n^3)$ time.

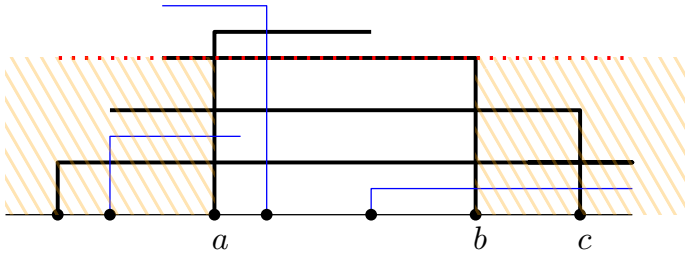

Figure 5: Illustration for the dynamic programming for two-sided L-shapes. The clique Q is shown in black.

3.2 Computing a Clique in $O((\frac{n \log n}{\log \log n})^2)$ time

We now show how the implementation of the dynamic programming algorithm can be done faster by using the range maxima data structure. We first describe the data structure in Section 3.2.1 and then describe the details of the implementation of the dynamic program in Section 3.2.2.

3.2.1 Data Structure for Range Maxima Queries

Let P be a set of points in $\mathbb{R}^2$, where each point is assigned a positive weight. An *orthogonal range maxima query* on P receives an axis-aligned rectangle and returns a point with the maximum weight within the given rectangle. We assume that points are in rank space, i.e., each coordinate of a point is replaced by its rank in the set of corresponding coordinates.

For small point sets in a rank space, Nekrich [20] described a data structure that supports queries and weight updates in $O(1)$ time, as stated in the following lemma. Here we briefly outline the idea. Since the points are in rank space, the coordinates and weights can be represented using $\log n$ bits. Therefore, a set of $t = \frac{\log n}{6 \log \log n}$ weighted points can be described in a bit string of length $3t \log \log n = \frac{1}{2} \log n$ bits. Hence the solution to all possible $O(\log^2 n)$ queries for all possible sets that can arise from at most t points can be precomputed and stored in a table of size $O(2^{\frac{1}{2} \log n} \log^2 n) \in O(\sqrt{n} \log^2 n)$. Hence a query can be answered in constant time, and a weight update can be done in constant time by maintaining a pointer that keeps track of the set that represents the current set of points. Nekrich [20] showed how this idea can be leveraged to process a set of $O(\log^{2\varepsilon} n)$ points, where $0 < \varepsilon < 1$, using a 'grid approach' [1, 7].

Lemma 2 (Nekrich [20]) *If a point set P contains $O(\log^{2\varepsilon} n)$ points, where $0 < \varepsilon < 1$, and coordinates and weights of all points are bounded by $O(\log n)$, then we can support range maxima queries under weight updates on P in $O(1)$ time. The data structure uses a universal look-up table of size $o(n)$.*

For the general point set, we modify the range tree data structure to obtain an $O((\frac{\log n}{\log \log n})^2)$ time query and weight update. A range tree on a set of points in $\mathbb{R}^2$ is a balanced binary search tree on one coordinate of the points, where each node u stores a balanced binary search tree on the other coordinate over the points that are stored at the leaves of the subtree rooted at u. Consequently, a range tree consists of a first-level tree and a set of second-level trees stored at the nodes of the first-level tree. We assume the readers are familiar to the range tree data structure and refer to [10] for the detailed construction and properties of a range tree. We will leverage a generalization of the range tree data structure [20] that uses search trees of higher degree.

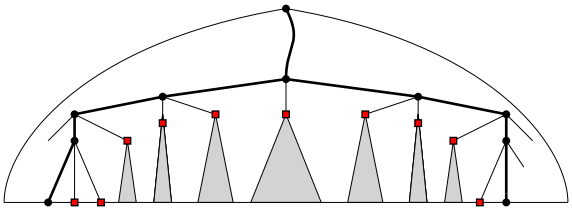

Figure 6: Illustration for the search paths in a balanced search tree of degree 3. The candidate nodes are shown in red squares.

Lemma 3 *Let P be a set of n points with all the coordinates and weights bounded by $O(n)$. Then there exists a data structure that supports range maxima queries and weight updates on P in $O((\frac{\log n}{\log\log n})^2)$ time.*

Proof: We construct a range tree using balanced search trees of degree $\log^\varepsilon n$, where $0 < \varepsilon \leq 1/2$. By T_y we denote the first-level tree which is constructed using y-coordinates. For a node w in T_y, we use T_x^w to denote the second-level tree stored at w.

A query (or an update) finds two search paths in T_y and thus $O(\frac{\log n}{\log\log n})$ nodes in T_y. At each such node w, we need to query T_x^w to identify $O(\frac{\log n}{\log\log n})$ nodes on the search paths in T_x^w. Hence we now have $O((\frac{\log n}{\log\log n})^2)$ candidate nodes in the second-level trees.

There are $O(\frac{\log n}{\log\log n})$ nodes in a search path, and each node may have $O(\log^\varepsilon n)$ children that contain points with correct coordinates. We refer to these children as candidate nodes (Figure 6). Therefore, if we simply store the maximum weight of a subtree at the root of the subtree, then to answer a query we need to take the maximum over all the $O((\frac{\log^{1+\varepsilon} n}{\log\log n})^2)$ candidate nodes in the second-level trees. To obtain an upper bound of $O((\frac{\log n}{\log\log n})^2)$ for query (or update) time, for each node w that belongs to a search path in T_y we find the search paths in the second-level tree T_x^w and query each node on the search path of the second-level tree in $O(1)$ time. The goal here is to leverage the data structure of Lemma 2, which requires the point coordinates to be bounded by $O(\log n)$ and the point set size to be bounded by $O(\log^{2\varepsilon} n)$. To satisfy this property at each node of the second-level tree, we take the following two steps at the time we construct T_x^w.

First, at each second-level tree T_x^w, we modify the y-coordinates of the points so that they are bounded by $O(\log^\varepsilon n)$. To achieve this, for each point p, we find the child of w that stores p. If p is stored at the jth child of w, then we replace the y-coordinate of p with j. Since the degree is bounded by $O(\log^\varepsilon n)$, the y-coordinates of the points of the second-level trees are now bounded by $O(\log^\varepsilon n)$. Second, at each node u of T_x^w, we select for each child of u, the points with the maximum weight for each y-coordinate. Since there are $O(\log^\varepsilon n)$ distinct y-coordinates and $O(\log^\varepsilon n)$ children, the number of points that we select is $O(\log^{2\varepsilon} n)$. Note that the y-coordinates of these points are already bounded by $O(\log n)$. We map the x-coordinate of each selected point q to the index of the child of u that stores q. Hence the x-coordinates become bounded by $O(\log^\varepsilon n)$. Since there are at most $O(\log^{2\varepsilon} n)$ points, the weight can be reduced to a rank space. Since $\varepsilon \leq 1/2$, the rank is bounded by $O(\log n)$. Since the number of selected points is $O(\log^{2\varepsilon} n)$ and the point coordinates are bounded by $O(\log n)$, we can store these points in a data structure of Lemma 2.

For each of the $O((\frac{\log n}{\log\log n})^2)$ queries at the second-level trees, we can now use the data structure of Lemma 2. Similarly, when the weight of a point p needs to be updated, we consider the $O((\frac{\log n}{\log\log n})^2)$ candidate nodes in the second level tree and for each node, pass the point coordinates

and weight information in the rank space to the data structure of Lemma 2 stored in the node. Hence the overall running time becomes $O((\frac{\log n}{\log\log n})^2)$. □

3.2.2 Implementation of the Dynamic Program

We now describe the implementation of the dynamic program using the data structure of Lemma 3. A straightforward implementation of the data structure takes $O(n^2)$ time. Assume that we are computing an entry $D(a, b)$ where a and b are the first and second highest L-shapes, respectively. For each L-shape v, we maintain a data structure S_v that maintains a set of weighted points as follows. Each point $p \in S_v$ corresponds to an L-shape w that intersects v. The x and y-coordinates of p are the x and y-coordinates of the bend point of w, and the weight of p is the solution of $D(v, w)$ or $D(w, v)$ depending on whether v precedes w on the ground line.

Let r and r' be two points on the ground line such that all the fixed ends of the L-shapes are between r and r'. To compute $D(a, b)$, it now suffices to compute two range maxima queries (Figure 5), as follows. One range is defined by the vertical segment of b, the horizontal line determined by the highest point of b, the ground line, and the vertical line through r'. The other range is defined by the vertical segment of a, the horizontal line determined by the highest point of b, the ground line, and the vertical line through r.

Since a range maxima query takes $O((\frac{\log n}{\log\log n})^2)$ time (Lemma 3), the dynamic programming table can be filled in $O(n^2(\frac{\log n}{\log\log n})^2)$ time. The following theorem summarizes the result of this section.

Theorem 3 *Given a set of n grounded two-sided L-shapes, one can compute a maximum clique in the corresponding intersection graph in $O((\frac{n\log n}{\log\log n})^2)$ time.*

4 Two-sided Square L-shapes

In this section, we consider two-sided square L-shapes, and give an $O(n^2 \log\log n)$-time algorithm to compute a maximum clique. We assume the L-shapes are in general position.

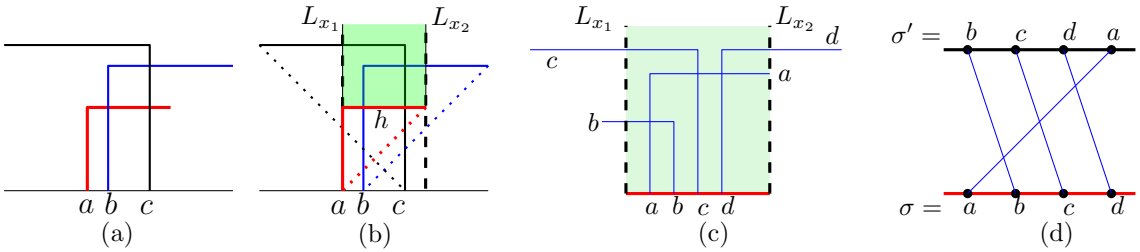

Figure 7: (a) A maximum clique in a two-sided square L-shape representation. (b) Illustration for the properties of the clique. (c)–(d) Construction of the permutation graph.

We first discuss some geometric properties of a maximum clique, which will help us to design the dynamic programming (Figure 7). Let a be the lowest L-shape of a maximum clique Q. Then all other L-shapes must intersect the horizontal segment h of a. Let L_{x_1} and L_{x_2} be vertical lines through the left and right end points of h. We now have the following observation.

Lemma 4 *Every square L-shape that intersects h, must intersect L_{x_1} or L_{x_2}.*

Proof: Suppose for a contradiction that b is an L-shape that intersects h but does not intersect L_{x_1} or L_{x_2}. Then the length of the horizontal segment of b is at most the length of h. Thus the maximum length for the vertical segment of b is also bounded by the length of h. Since the bend-point of b is above h, the vertical segment of b cannot reach the ground line, which contradicts that b is a grounded square L-shape. □

Let R be the region above h, bounded by L_{x_1} and L_{x_2}. Let H be the intersection graph induced by the L-shapes intersecting h. In the following we show that H is a permutation graph. Here a *permutation graph* is an intersection graph of straight line segments where the line segments are drawn between a pair of parallel lines, i.e., all the top end points of these segments lie on one line and the bottom end points lie on the other.

Lemma 5 *The graph H is a permutation graph.*

Proof: Let σ be the labels of the L-shapes that intersect h in left to right order, and let σ' be the labels of these L-shapes in the clockwise order of their intersection points on the boundary of R excluding h. Let H' be the permutation graph determined by the orderings σ and σ' on two parallel lines, as illustrated in Figure 7(c)–(d). It now suffices to prove that two vertices are adjacent in H (i.e., the corresponding L-shapes intersect) if and only if they are adjacent in H' (i.e., the corresponding line segments intersect).

Let a and b be two L-shapes that intersect in the square-L representation of H. If they both intersect L_{x_1} (resp., L_{x_2}), then their ordering in σ is different from that of σ'. If one intersects L_{x_1} and the other intersects L_{x_2}, then again, their ordering in σ is different from that of σ'. Thus in both cases, the corresponding vertices in H' must be adjacent, i.e., the line segments must intersect in the permutation graph representation of H'.

Consider now the case when the L-shapes a and b do not intersect. If they both intersect L_{x_1} (resp., L_{x_2}), then one L-shape must 'nest' the other and thus their ordering would be the same in σ and σ'. If one L-shape intersects L_{x_1} and the other intersects L_{x_2}, then the one that intersects L_{x_1} must appear to the left of the other on the ground line. Therefore, in both cases, the corresponding vertices in H' cannot be adjacent, i.e., the line segments cannot intersect in the permutation graph representation of H'. □

We precompute a permutation representation of the permutation graph for every L-shape. A maximum clique in a permutation graph corresponding to an L-shape determines a maximum clique containing that L-shape. Since a maximum clique in a permutation graph can be obtained in $O(n \log\log n)$ time [25], we can find the maximum clique containing an L-shape in the same time.

Finally, we iterate the above process over all L-shapes to find the maximum clique. Let S be the input L-shapes. It is straightforward to precompute the permutation representation of the permutation graph for every L shape in $O(n^2)$ time in total by first computing the sorted orders of the vertical segments and horizontal segments of the L-shapes, and then for each segment a, computing the L-shapes that intersects h, L_{x_1} and L_{x_2} in sorted order. Hence the time complexity for computing maximum clique is $O(n^2) + \sum_{q \in S} O(n \log\log n) \in O(n^2 \log\log n)$, where S is the set of L-shapes in the input.

The following theorem summarizes the result of this section.

Theorem 4 *Given a set of n grounded two-sided square L-shapes, one can find the maximum clique of the corresponding intersection graph in $O(n^2 \log\log n)$ time.*

5 One-sided L-shapes

In this section, we consider 1-sided L-shapes, and give an $O(n^2 \log\log n)$-time algorithm to compute a maximum clique. We assume the L-shapes are in general position. We noticed that Chmel [9] has independently discovered this result at the same time the conference version of this paper was accepted. Chmel mentioned an $O(n^2 \log n)$ time complexity. While the algorithm is the same as ours, in our analysis we use an improved subroutine for computing a maximum clique in a restricted class of graphs [25] to obtain a faster overall running time.

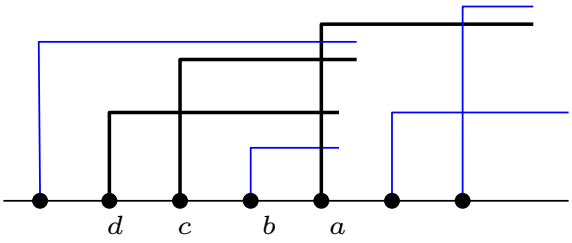

Figure 8: Illustration for the dynamic programming for 1-sided L-shapes. The clique Q is shown in black.

Let S be the set of L-shapes in the input and let Q be a maximum clique. Let a be the highest L-shape in Q. Then all the other L-shapes in Q must intersect the vertical segment of a (Figure 8). There may also exist L-shapes (e.g., b) that do intersect the vertical segment of a but do not belong to Q.

Let $D(S')$ be the maximum clique in the intersection graph of the set $S' \subseteq S$. For any L-shape $q \in S$, let $N(q)$ be the subset of S such that every L-shape in $N(q)$ intersects the vertical segment of q. Then $D(S)$ can be defined as follows.

$$D(S) = \max_{q \in S}(D(N(q)) + 1)$$

To compute the maximum clique efficiently, we do some preprocessing. We first compute the adjacency list of the intersection graph G of S in $O(n^2)$ time. We then compute a sorted list S_x consisting of the fixed end points (on the grounded line) of all the L-shapes. We next compute another sorted list S_h of the heights of the L-shapes (Figure 9). Both these lists can be computed in $O(n \log n)$ time. Hence the total preprocessing takes $O(n^2) + O(n \log n) \in O(n^2)$ time.

For each $q \in S$, we now compute the maximum clique in $N(q)$. First note that $N(q)$ corresponds to a permutation graph, where the edges are determined by the intersection of the L-shapes in $N(q)$. We first find two ordered lists for the L-shapes of $N(q)$, one list corresponds to the order they appear on the ground line, and the other list corresponds to the order they appear on the vertical segment of q. We then use an $O(n \log\log n)$-time algorithm [25] for computing a maximum clique in a permutation graph to compute the maximum clique in $N(q)$.

To list the L-shapes of $N(q)$ in the order they appear on the ground line, we scan S_x from left to right and create a new ordered list S'_x with the L-shapes that intersect the vertical segment of q. We then scan S_h and create a new ordered list S'_h that contains the L-shapes intersecting the vertical segment of q. Constructing S'_x and S'_h takes $O(n)$ time. Hence computing $N(q)$ and finding a maximum clique in $N(q)$ takes $O(n) + O(n \log\log n)$ time. Thus the total running time

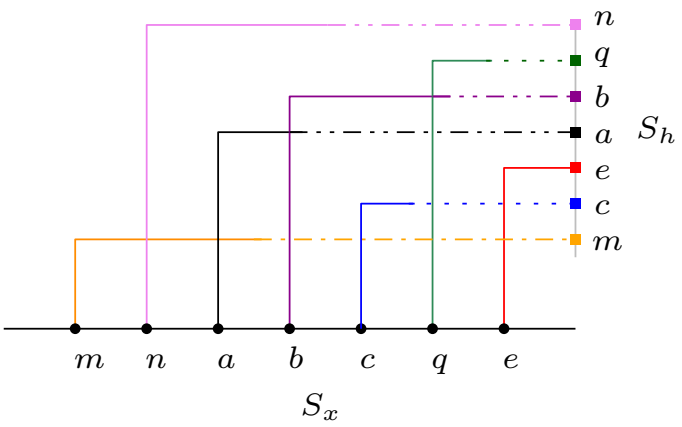

Figure 9: Illustration for S_x and S_h.

is

$$\sum_{q \in S} O(n) + O(n \log\log n) \in O(n^2 \log\log n).$$

The following theorem summarizes the result of this section.

Theorem 5 *Given a set of n grounded 1-sided L-shapes, one can find the maximum clique of the corresponding intersection graph in $O(n^2 \log\log n)$ time.*

6 Separation Between Graph Classes

In this section we show that there exist graphs that can be represented with 2-sided square L-shapes, but not with grounded 1-sided L-shapes (Theorem 6). Furthermore, there exist graphs that can be represented with grounded 2-sided L-shapes but not with grounded 2-sided square L-shapes (Theorem 7).

Theorem 6 *There is a graph G which admits an intersection graph representation with 2-sided grounded square L-shapes but not with 1-sided grounded L-shapes.*

Proof: Jelínek and Töpfer [12] showed that there exists a graph G that can be represented using grounded 2-sided L-shapes but not with grounded 1-sided L-shapes. It thus suffices to show that G admits an intersection representation with grounded 2-sided square L-shapes.

Figure 10(a) shows an intersection graph representation of G with grounded 2-sided L-shapes. The L-shapes can be partitioned into five groups as follows:

Group 1: Four L-shapes x_1, x_2, x_3, x_4 with edges $(x_1, x_4), (x_1, x_2), (x_2, x_3)$.

Group 2: Four L-shapes x_5, x_6, x_7, x_8 with edges $(x_5, x_8), (x_7, x_8), (x_6, x_7)$.

Group 3: Four L-shapes $x_9, x_{10}, x_{11}, x_{12}$ similar to Group 1.

Group 4: Four L-shapes $x_{13}, x_{14}, x_{15}, x_{16}$ similar to Group 2.

Group 5: A cycle $y_1, \ldots, y_{80}$ with x_i intersecting the L-shape corresponding to y_{5i}.

The L-shapes corresponding to Group 1 and 3 are shown in blue. The L-shapes corresponding to Group 2 and 4 are shown in red. The L-shapes corresponding to the cycle are shown in black.

Figure 10(b) and (c) illustrate a representation of Group 1 and Group 2 with square L-shapes and their associated L-shapes on the cycle. We can scale up and patch the representation of Group 2 with that of Group 1 such that the rightmost L-shape intersects the leftmost L-shape of Group 2. We can thus extend the drawing to integrate the Groups 3 and 4. A schematic representation of this configuration is illustrated in Figure 10(d). The first and the last L-shapes on the cycle (i.e., y_1 and y_2) only intersect the leftmost and rightmost of these configuration. Hence it is straightforward to add them to the configuration completing the square L-shape representation of G. □

We now prove that there exists a 2-sided grounded L-shape graph that is not a 2-sided grounded square-L graph. We first provide a remark that would help to present this result.

Remark 1 *Let p, q, r be three grounded square-L shapes in this left to right order on the ground line. Then the following two properties hold.*

A. If r is lower than p and both turn clockwise, then q cannot intersect p and r simultaneously.

B. If r turns clockwise and p turns anticlockwise, then q cannot intersect p and r simultaneously.

Theorem 7 *There exists a graph G that admits an intersection graph representation with 2-sided grounded L-shapes but not with 2-sided grounded square L-shapes.*

Proof: We will use the wheel graph G of 6 vertices to prove this theorem (Figure 11(a)). G has an intersection representation with 2-sided grounded L-shapes [3]. Suppose for a contradiction that G admits an intersection representation R with grounded 2-sided square L-shapes. Consider now a representation R' of the 5-cycle $C = (a, b, c, d, e)$ in R. We now examine the first and last L-shapes of R', and examine two cases depending on whether they are adjacent on the 5-cycle.

Case 1 (The first and last L-shapes in R' are adjacent in C): Without loss of generality assume that a and e are the leftmost and rightmost L-shapes, and a is taller than e. Since e is adjacent to a but not to b, b must be lower than e. Since b is adjacent to a, it must be of type ⅂. We now consider two subcases depending on c's position on the ground line.

Case 1.1 (c is lies to the right of b): Since c is adjacent to b, it must be of type ⅂. Since d needs to intersect c and e, d must lie to the left of c on the ground line. Since d must not intersect b it needs to lie to the right of b. Hence a, b, c, d and e have unique x-coordinate order on the ground line, as illustrated in Figure 11(b).

We now need to add the vertex x.

If we put x between a and b, then to intersect both a and e, it must be higher than e. First assume that the vertical segment of d intersects e. If a is of type ⅂, then d must be of type ⅂ and thus d must be lower than a. By Remark 1 property A, x would fail to intersect both a and d simultaneously. If a is of type ⅂ and d is of type ⅂, then by Remark 1 property B, x would fail to intersect both a and d simultaneously. If a and d are both of type ⅂, then a must be lower than d and by Remark 1 property A, x would fail to intersect both a and d simultaneously. Assume now that the horizontal segment of d intersects e. In this scenario, d is lower than a and is of type ⅂. Hence, by Remark 1, x fails to intersect both a and d simultaneously.

If we put x between b and e, then by Remark 1 property A, x cannot intersect both b and e simultaneously.

Therefore, x either lies to the right of e or to the left of a. Consider first that x lies to the left of a. Since c does not intersect a and e, it is inside the region bounded by a and e. Hence x must be lower than c and intersect the vertical segment of c. But then x would need to lie to the right of

a. Consider now the case when x lies to the right of e. In this case the height of x must be smaller than that of c. Since c does not intersect a, in the square representation x will fail to intersect a.

Case 1.2 (c is lies to the left of b): Similar to Case 1.2, this case also results in a unique x-coordinate ordering for the L-shapes on the 5-cycle. If c is of type ⅂, then to intersect b it has to be higher than b. But then c would also intersect a. Since c is not adjacent to a, c must be of type ⅃. Since d is not adjacent to b, it must be between b and e. Furthermore, d must be of type ⅃ to intersect c and e simultaneously. Figure 11(c) illustrates this scenario.

If x lies to the left of c, then the height of x must be smaller than c. Since c does not intersect e, x will also fail to intersect e.

If x lies between c and d, then first assume that the vertical segment of d intersects e. If a is of type ⅃, then d must be of type ⅃ and thus d must be lower than a. By Remark 1 property A, x would fail to intersect both a and d simultaneously. If a is of type ⅂ and d is of type ⅃, then by Remark 1 property B, x would fail to intersect both a and d simultaneously. If a and d are both of type ⅂, then a must be lower than d and by Remark 1 property A, x would fail to intersect both a and d simultaneously. Assume now that the horizontal segment of d intersects e. In this scenario, d is lower than a and is of type ⅃. Hence, by Remark 1, x fails to intersect both a and d simultaneously.

If x lies to the right of d, then to intersect both b and e, x must lie also to the right of e. In this case the height of x must be smaller than c. Hence x will fail to intersect a.

Case 2 (The first and last L-shapes in R' are not adjacent in C): Without loss of generality assume that a and c are the first and last L-shapes, and a is taller than c.

If a and c are both of type ⅃, then since b appears between them, by Remark 1 property A, b cannot intersect both a and c simultaneously.

If a is of type ⅂ and c is of type ⅃, then by Remark 1 property B, b cannot intersect both a and c simultaneously.

We now consider the scenario when c is of type ⅂. If b intersects the vertical segment of c, then b would be lower than c, and thus b would be lower than a. Therefore, b would fail to intersect a. We may thus assume that b must intersect the horizontal segment of c. We now consider two cases depending on whether the vertical or horizontal segment of b intersects a.

Case 2.1 (The horizontal segment of b intersects a): If d lies to the left of b, then it would lie in the region bounded by b and intersect the horizontal segment of c. This situation is illustrated in Figure 12(a). To intersect d and a simultaneously, e must lie to the left of d. We now consider the placement of x. If x lies to the right of d, then to intersect d, it must be lower than d. Since d does not intersect a, x would fail to intersect a. If x lies to the left of d and intersects the vertical segment of c, then x will be lower than c and nested inside c, and hence, would fail to intersect e. Therefore, the only remaining option for x is to lie to the left of d and to intersect the horizontal segment of c. Since the horizontal segment of b intersects a, b is of type ⅂. Since e intersects a but not b, e is of type ⅂. Therefore, by Remark 1 property A, x cannot intersect b and e simultaneously.

Consider now the scenario when d lies to the right of b. Since e needs to intersect a, but to avoid b and c, its bend must be in the region bounded by a and b. Since d lies to the right of b, to avoid the intersection with b, d must be higher than b. Therefore, d fails to intersect e.

Case 2.2 (The vertical segment of b intersects a): Recall that we have already argued b to intersect the horizontal segment of c. This scenario is illustrated in Figure 12(b).

If d lies to the left of b, then to avoid the intersection with a and b, d must lie in the region bounded by a and b. To intersect d and a simultaneously, e must lie to the left of d. We now consider the placement of x. If x lies to the right of d, then to intersect d, it must be lower than d. Since d does not intersect a, x would fail to intersect a. If x lies to the left of d and intersects the

vertical segment of c, then x will be nested inside c and would fail to intersect e. Therefore, the only feasible option for x is to lie to the left of d and to intersect the horizontal segment of c, x must lie to the right of e. If x intersects the vertical segment of a, then it fails to intersect b. Therefore, x must be higher than a. We now show that x cannot intersect e and b simultaneously. First consider the scenario when b is of type $\urcorner$. Since e lies between a and b, to avoid an intersection with b, e must be lower than b. Hence by Remark 1 property A, x fails to intersect e and b simultaneously. Consider now the scenario when b is of type Γ. If e intersects the vertical segment of a, then by Remark 1 property B, x fails to intersect e and b simultaneously. If e intersects the horizontal segment of a, then to avoid an intersection with b, e can either be of type $\urcorner$, or be of type Γ with a height larger than b. Hence by Remark 1, x fails to intersect e and b simultaneously.

Consider now the scenario when d lies to the right of b (Figure 12(c)). If e lies to the left of b and b is of type $\urcorner$, then to avoid the intersection with b, e must be lower than b. Hence e would fail to intersect d. Consider now the scenario when e lies to the left of b and b is of type Γ. Since b must intersect c (while c is to the right of d and d does not intersect b), d must be lower than b. Hence e fails to intersect d. Finally, consider the scenario when e lies to the right of b. Since c is the rightmost L-shape, e must be to the left of c. Furthermore, e must be lower than c to avoid any intersection with b and c. Since a is taller than c, e fails to intersect a. □

7 Conclusion

In this paper we examined the maximum clique problem for grounded 1-bend string graphs. We showed the problem to be APX-hard for y-monotone strings. We also showed that the problem remains APX-hard when we relax monotonicity, but restrict the bends and free end points of the strings to lie on three horizontal lines. The most intriguing open problem in this context is to settle the time complexity for grounded segment graphs.

Open Problem 1: Does there exist a polynomial-time algorithm to compute a maximum clique in grounded segment graphs?

There are also various questions worth investigating for grounded 1-bend strings where the bends and end points lie on a few lines. If we allow only one horizontal line, then the resulting strings become grounded segments and the corresponding intersection graph is a permutation graph, where one can find a maximum clique in polynomial time. For a fixed number of lines the problem is polynomial-time solvable for segments [18]. Therefore, it would be interesting to examine whether the problem becomes polynomial-time solvable for two horizontal lines, where we can explore 1-bend strings. We think such restriction on the number of lines would be non-trivial even for strictly y-monotone 1-bend strings.

Open Problem 2: Does there exist a polynomial-time algorithm to compute a maximum clique in grounded 1-bend string graphs when the bends and end points are restricted to lie on two horizontal lines above the ground line?

We developed polynomial-time algorithms to find maximum clique for various types of L-shapes. A natural question is whether the running times of these algorithms can be improved. It would also be interesting to find non-trivial lower bounds on the time complexity.

Open Problem 3: Does there exist a subquadratic-time algorithm to compute a maximum clique in grounded 1-sided or 2-sided L-shape graphs?

We examined the inclusion relations among various types of grounded L-shape graphs. However, the inclusion relations between grounded L-shape graphs and grounded 1-bend string graphs (i.e., non-axis aligned setting) are not well understood.

Open Problem 4: Find a tight function $f(n)$ such that every grounded 2-sided L-shape graph with n vertices can be drawn as an intersection graph of grounded 1-bend strings with the bends and end points on at most $f(n)$ horizontal lines.

Acknowledgements

The research of Debajyoti Mondal is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC). We thank anonymous reviewers for their careful reading, insightful comments and suggestions.

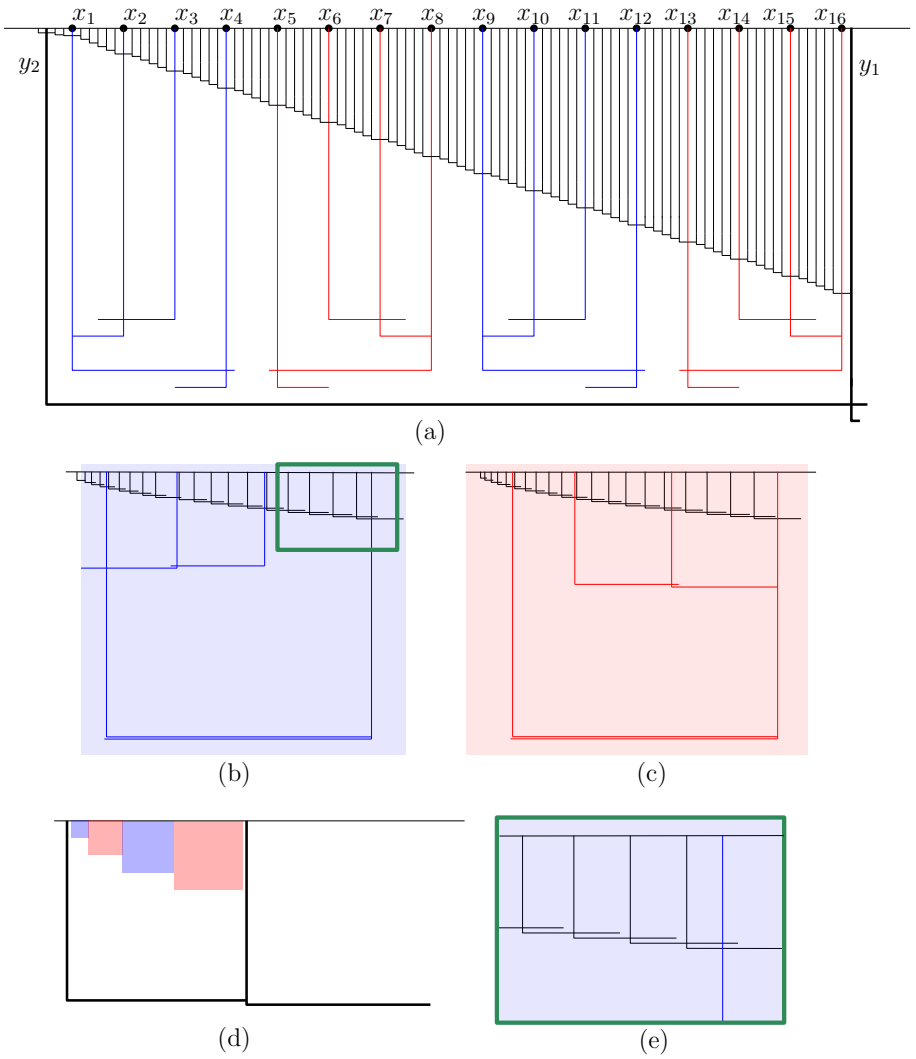

Figure 10: (a) An intersection graph G of grounded 2-sided L-shapes that cannot be represented with 1-sided L-shapes. (b)–(c) Illustration for an intersection representation of G with grounded 2-sided square L-shapes. (e) A zoomed in view of the L-shapes of figure (b).

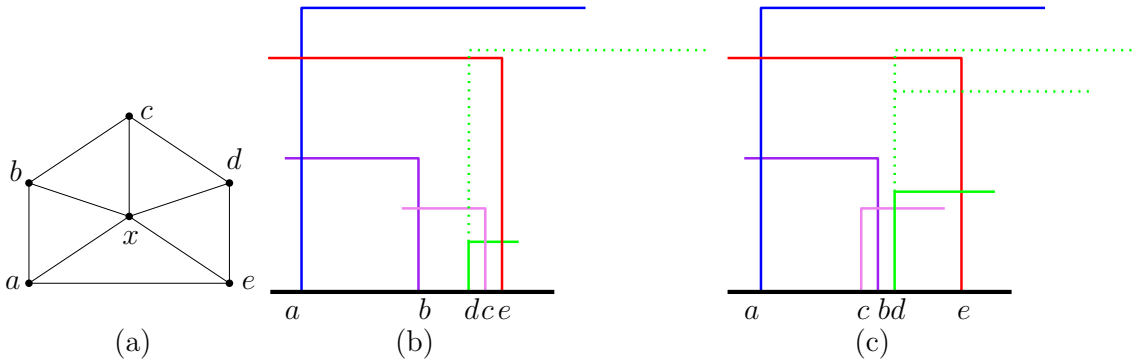

Figure 11: (a) A wheel graph. Illustration for (b) Case 1.1 (c) and Case 1.2.

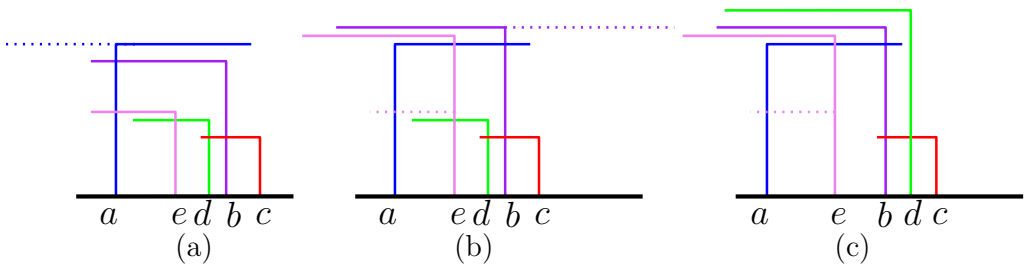

Figure 12: Illustration for (a) Case 2.1, and (b)–(c) Case 2.2.

References

[1] S. Alstrup, G. S. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 198–207. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892088.

[2] S. Benzer. On the topology of the genetic fine structure. *Proceedings of the National Academy of Sciences of the United States of America*, 45(11):1607, 1959. doi:10.1073/pnas.45.11.1607.

[3] P. Bose, P. Carmi, J. M. Keil, A. Maheshwari, S. Mehrabi, D. Mondal, and M. Smid. Computing maximum independent set on outerstring graphs and their relatives. *Comput. Geom.*, 103:101852, 2022. doi:10.1016/j.comgeo.2021.101852.

[4] S. Cabello, J. Cardinal, and S. Langerman. The clique problem in ray intersection graphs. In *European Symposium on Algorithms*, pages 241–252. Springer, 2012. doi:10.1007/s00454-013-9538-5.

[5] S. Cabello, K. Jain, A. Lubiw, and D. Mondal. Minimum shared-power edge cut. *Networks*, 75(3):321–333, 2020. URL: https://doi.org/10.1002/net.21928.

[6] J. Cardinal, S. Felsner, T. Miltzow, C. Tompkins, and B. Vogtenhuber. Intersection graphs of rays and grounded segments. *J. Graph Algorithms Appl.*, 22(2):273–295, 2018. doi:10.7155/jgaa.00470.

[7] T. M. Chan, K. G. Larsen, and M. Patrascu. Orthogonal range searching on the ram, revisited. In F. Hurtado and M. J. van Kreveld, editors, *Proceedings of the 27th ACM Symposium on Computational Geometry (SoCG)*, pages 1–10. ACM, 2011. doi:10.1145/1998196.1998198.

[8] M. Chlebík and J. Chlebíková. The complexity of combinatorial optimization problems on d-dimensional boxes. *SIAM J. Discret. Math.*, 21(1):158–169, 2007. doi:10.1137/050629276.

[9] P. Chmel. Algorithmic aspects of intersection representations, June 2020. Bachelor Thesis, Charles University.

[10] M. T. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational geometry: algorithms and applications*. Springer Science & Business Media, 2000. doi:10.1007/978-3-540-77974-2.

[11] J. Fox and J. Pach. Computing the independence number of intersection graphs. In D. Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1161–1165. SIAM, 2011. doi:10.1137/1.9781611973082.87.

[12] V. Jelínek and M. Töpfer. On grounded l-graphs and their relatives. *Electr. J. Comb.*, 26(3):P3.17, 2019. URL: https://www.combinatorics.org/ojs/index.php/eljc/article/view/v26i3p17.

[13] J. M. Keil, J. S. Mitchell, D. Pradhan, and M. Vatshelle. An algorithm for the maximum weight independent set problem on outerstring graphs. *Computational Geometry*, 60:19–25, 2017. doi:10.1016/j.comgeo.2016.05.001.

[14] J. M. Keil, D. Mondal, and E. Moradi. Finding a maximum clique in a grounded 1-bend string graph. In J. M. Keil and D. Mondal, editors, *Proceedings of the 32nd Canadian Conference on Computational Geometry (CCCG)*, pages 160–166, 2020.

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

[16] J. Matoušek. Intersection graphs of segments and $\exists\mathbb{R}$. *arXiv preprint arXiv:1406.2636*, 2014.

[17] T. A. McKee and F. R. McMorris. *Topics in intersection graph theory*. SIAM, 1999.

[18] M. Middendorf and F. Pfeiffer. The max clique problem in classes of string-graphs. *Discret. Math.*, 108(1-3):365–372, 1992. doi:10.1016/0012-365X(92)90688-C.

[19] N. Nash and D. Gregg. An output sensitive algorithm for computing a maximum independent set of a circle graph. *Inf. Process. Lett.*, 110(16):630–634, 2010. doi:10.1016/j.ipl.2010.05.016.

[20] Y. Nekrich. Four-dimensional dominance range reporting in linear space. In *36th International Symposium on Computational Geometry, SoCG 2020, June 23-26, 2020, Zürich, Switzerland*, pages 59:1–59:14, 2020. (full version: https://arxiv.org/abs/2003.06742). doi:10.4230/LIPIcs.SoCG.2020.59.

[21] M. Pergel and P. Rzążewski. On edge intersection graphs of paths with 2 bends. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 207–219. Springer, 2016. doi:10.1016/j.dam.2017.04.023.

[22] F. W. Sinden. Topology of thin film RC circuits. *Bell System Technical Journal*, 45(9):1639–1662, 1966. doi:10.1002/j.1538-7305.1966.tb01713.x.

[23] R. Tamassia. *Handbook of graph drawing and visualization*. CRC press, 2013.

[24] A. Tiskin. Fast distance multiplication of unit-monge matrices. *Algorithmica*, 71(4):859–888, 2015. doi:10.1007/s00453-013-9830-z.

[25] M. Yu, L. Tseng, and S. Chang. Sequential and parallel algorithms for the maximum-weight independent set problem on permutation graphs. *Inf. Process. Lett.*, 46(1):7–11, 1993. doi:10.1016/0020-0190(93)90188-F.