
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 11, no. 1, pp. 145–164 (2007)

Fixed-Location Circular Arc Drawing of Planar Graphs*†

Alon Efrat *Cesim Erten* *Stephen G. Kobourov*

Department of Computer Science
University of Arizona
alon@cs.arizona.edu cesim@cs.arizona.edu kobourov@cs.arizona.edu

Abstract

In this paper we consider the problem of drawing a planar graph using circular arcs as edges, given a one-to-one mapping between the vertices of the graph and a set of points in the plane. If for every edge we have only two possible circular arcs, then a simple reduction to 2SAT yields an $O(n^2)$ algorithm to find out if a drawing with no crossings can be realized, where n is the number of vertices in the graph. We present an improved $O(n^{7/4} polylog\ n)$ time algorithm for this problem. For the special case where the possible circular arcs for each edge are of the same length, we present an even more efficient algorithm that runs in $O(n^{3/2} polylog\ n)$ time. We also consider two related optimization versions of the problem. First we show that minimizing the number of crossings is NP-hard. Second we show that maximizing the number of edges that can be realized without crossings is also NP-hard. Finally, we show that if we have three or more possible circular arcs per edge, deciding whether a drawing with no crossings can be realized is NP-hard.

Article Type	Communicated by	Submitted	Revised
Regular paper	M. Kaufmann	November 2004	February 2006

* This work was partially supported by the NSF under grant ACR-02229290.

† A preliminary version of this paper appeared in Proceedings on Graph Drawing (GD'03).

1 Introduction

One of the main goals in information visualization is to present the given information in a clear and concise manner. Often, the information is captured in the form of a graph, where the vertices represent a set of objects and the edges represent a relationship between the objects. The quality of a graph layout is typically measured by aesthetic criteria, such as, smooth edges, few edge crossings, uniform distribution of the vertices, and display of inherent symmetries.

Depending on the data source, the graph may or may not contain knowledge about the relative positions of the vertices for a 2 dimensional or 3 dimensional layout. In telephone graphs, where vertices are phone numbers and edges represent phone calls, natural 2D coordinates for the vertices can be assigned based on the physical location of the corresponding phones. Similarly, knowledge about the positions of vertices is contained in airline graphs, where vertices correspond to cities and edges correspond to flights between cities. Standard graph drawing algorithms, such as spring embedders [13] and Sugiyama-style layered methods [9] cannot handle such additional knowledge and there does not seem to be a straight-forward way to augment them.

Thus, a natural question that arises in graph drawing is whether a graph with fixed vertex locations can be drawn without crossings, when several choices are given for each of the edges. From an information visualization point of view convex edges are preferable, i.e., straight line segments or circular arcs. In general, embedding a planar graph at fixed locations and drawing it with straight lines may result in many crossings. Using circular arcs instead can reduce or eliminate the crossings; see Fig. 1(a). With this in mind, we address the problem of crossing-free drawings of graphs with predetermined vertex locations, using circular arcs.

We first consider the *2-Circular Arcs Drawing (2CAD)* problem, in which each edge has to be drawn as one of the two circular arcs defined by a circle passing through the endpoints. This problem is reminiscent of the *Manhattan wiring problem*. Consider the axis-aligned rectangle with a diagonal defined by the line segment between two vertices connected by an edge. Then the two semi-rectangles separated by the diagonal are the two choices for drawing the edge. This formulation of the problem can be decided in $O(n \log n)$ time, using an efficient *find and delete* data structure (find intersections between pairs of semi-rectangles and delete a semi-rectangle from the data structure) [16].

The same approach cannot be applied to the 2CAD problem directly. Although efficient data structures exist for operations on full circles, no such data structures exist for circular arcs or even semi-circles. The novelty of our 2CAD algorithm is that we provide a way to use an efficient data structure for full circles to solve the problem with circular arcs. For the sake of completeness, we first show that the 2CAD problem can be reduced to 2SAT and thus solved in $O(n^2)$ time. Then we present an $O(n^{7/4} polylog\ n)$ time algorithm for the general case of the 2CAD problem and an $O(n^{3/2} polylog\ n)$ time algorithm for the restricted case where all the circular arcs are half-circles. Although the practical gain in terms of running time is not significant, we believe that our approach

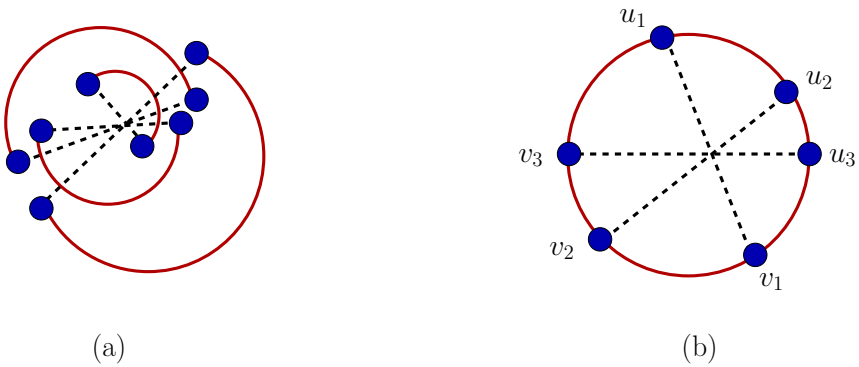


Figure 1: (a) An example graph with fixed vertex locations (edges shown as dashed segments) that requires $\Theta(n^2)$ crossings if its edges are drawn as straight-line segments and none if its edges are drawn as circular arcs; (b) A planar graph (edges shown as dashed segments) that cannot be drawn without crossings using *any* circular arc segments.

for solving the 2CAD problem might be of independent interest to solve similar problems.

Next, we consider the *3-Circular Arcs Drawing (3CAD)* problem, where for each edge there are three circular arcs to choose from. We show that the 3CAD problem is NP-hard. Finally, we consider two optimization problems related to the 2CAD problem. Although using circular arcs to represent the edges allows a certain flexibility, not every planar graph can be drawn without crossings using circular arcs. Fig. 1(b) shows an example of a planar graph that cannot be drawn without crossings using *any* circular arc segments. This difficulty suggests two optimization problems: *Min2CAD* is the problem of minimizing the number of crossings for a given 2CAD instance by representing *every* edge with an appropriate circular arc. *Max2CAD* is the problem of maximizing the number of edges that can be drawn without crossings using circular arcs. We show that both of these optimization problems are NP-hard.

2 Previous Work

Several variations of the problem of embedding a planar graph at fixed point locations have been studied. If we can choose the mapping between the vertices V and the points P, then Kaufmann and Wiese [17] show that the graph can be drawn without crossings using 2 bends per edge in polynomial time. However, if the mapping between V and P is given, Pach and Wenger [21] show that $\Omega(n)$ bends per edge are necessary to guarantee planarity, where n is the number of vertices in the graph. Godau [14] shows that given a graph if each vertex is allowed to move slightly in the neighborhood of a fixed point then the problem of deciding whether there exists a plane drawing of the graph using straight-line edges is NP-hard.

Drawing graphs with circular arcs without assigned vertex locations has been considered by Cheng *et al.* [6] in the context of planarity, angular resolution and drawing area. The problems under consideration in this paper are also related to the *k-position point labeling problem*, extensively studied in *map labeling* [3, 23, 24]. In the k-position point labeling problem we are given a set of points and a set of k possible label positions for each point and we would like to find a labeling of the points that optimizes specific criteria. Criteria such as maximizing the number of labeled points [3] and maximizing the size of the labels [7, 8, 12] have been considered. A variant of the map labeling problem is reduced to 2SAT and NP-completeness results are presented in [12]. Although these problems are related to drawing planar graphs with circular arcs, there is a significant difference: whereas map labeling problems are restricted by region intersections, drawing planar graphs with circular arcs is restricted by circular arc intersections.

3 Circular Arcs Drawing: 2CAD

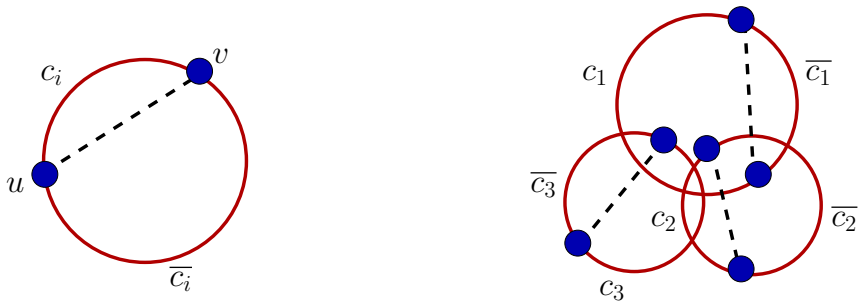
The input to the problem is a planar graph $G = (V, E)$, a point set P, a one-to-one function $f : V \to P$ such that $|V| = |P| = n$ and $|E| = m$, and a circle C_i for each edge $e_i = (u, v) \in E$ such that C_i passes through u, v . The two vertices, u and v, determine two circular arcs on C_i; let c_i and $\overline{c_i}$ be their labels; see Fig. 2(a). We would like to find out whether G can be drawn without crossings using c_i or $\overline{c_i}$ for each e_i, and if so, to provide such a drawing of G. Note that $m = O(n)$ as G is a planar graph.

We first suggest a straightforward solution of the problem using a reduction from 2CAD to 2SAT. The reduction to a 2SAT formula Φ requires that we identify all intersections between circular arcs. For each such intersection $c_i \cap c_j \neq \emptyset$, we add the clause $(\overline{c_i} \vee \overline{c_j})$ to Φ; see Fig 2(b). As there are $O(n^2)$ crossings, the reduction takes $O(n^2)$ time and results in a 2SAT formula Φ with $O(n)$ variables and $O(n^2)$ clauses. It is easy to see that G can be drawn without intersections if and only if the corresponding formula Φ is satisfiable. As 2SAT can be solved in time linear in the number of clauses and variables [4, 11], we have an $O(n^2)$ time algorithm for the 2CAD problem. However, we can do better.

3.1 The 2CAD Algorithm

Note that for a given edge $e_i = (u, v) \in E$ we consider as possible drawings of e_i the circular arcs defined by the circle C_i and the position of u and v on C_i. Alternatively, we could consider the axis-aligned rectangle having its diagonal as the line segment (u, v) and then consider the 2 semi-rectangles separated by the diagonal as two choices for drawings of the edge e_i. This formulation of the problem is known as the *Manhattan wiring problem* which can be efficiently solved in $O(n \log n)$ time [16].

We describe a new algorithm that solves a more general problem for circular



(a) (b)

Figure 2: (a) Given the circle C_i, edge $e_i = (u, v)$ is drawn either with the circular arc c_i or $\overline{c_i}$. (b) An example of a 2SAT reduction: $(\overline{c_1} \vee c_3) \wedge (\overline{c_1} \vee \overline{c_3}) \wedge (\overline{c_3} \vee \overline{c_2}) \wedge (\overline{c_1} \vee \overline{c_2}) \wedge (c_1 \vee c_2)$.

arcs. Our approach is different from the Manhattan wiring problem in that we perform operations only on complete circles. More formally, we *find/delete* complete circles as part of an intersection, as opposed to performing the operations on the circular arcs directly.

Let $\mathcal{D}$ denote the data structure used to store all the circles. Let $find(\mathcal{D}, c_i)$ be a function that finds a circle C_j intersecting circular arc c_i, and $delete(\mathcal{D}, C_i)$ be a function that deletes the circle C_i from $\mathcal{D}$. Let $\alpha(n), \beta(n)$ denote the time required to perform the *find* and *delete* operations, respectively. We next describe how to construct the data structure $\mathcal{D}$ and how to perform the *find/delete* operations efficiently. The main algorithm is shown on Fig. 3.

Let a *possible* circular arc be one that is not yet chosen and not yet discarded and let $\mathcal{P}$ denote the set of *possible* circular arcs. $\mathcal{P}$ initially contains all the circular arcs. In the *while* loop of line 1, we start with an arbitrary circular arc c_i and choose it to remain in $\mathcal{P}$. Then we traverse the circles in a depth-first manner and delete from $\mathcal{P}$ all circular arcs that are conflicting with the traversed arcs chosen to remain in $\mathcal{P}$ so far. Let $dfsnumber(c_i)$ denote the *dfsnumber* of c_i during this traversal. Note that *dfsnumber* is cumulative, in the sense that the number increases throughout the whole loop (lines 1-5). Each circle is found and deleted exactly once, resulting in a sequence of $O(n)$ *find/delete* operations and requiring $O(n \times (\alpha(n) + \beta(n))$ time. At the end of the *while* loop of line 1 in the main algorithm, the set $\mathcal{P}$ contains exactly one circular arc for each edge. However, $\mathcal{P}$ might contain intersections. Let $\mathcal{C}$ denote the set of *certain* circular arcs, ones that must definitely be in the final output. At this point we make the following observation:

Observation 1 *Let $c_i, c_j \in \mathcal{P}$ and $dfsnumber(c_i) < dfsnumber(c_j)$. At the end of the while loop of line 1, if $c_i \cap c_j \neq \emptyset$ then we must have $\overline{c_i} \in \mathcal{C}$.*

Algorithm 2CAD

1: **while** $\mathcal{D}$ not empty
2: **let** C_i be a circle in $\mathcal{D}$
3: *delete*$(\mathcal{D}, C_i)$, *delete* $\overline{c_i}$ from $\mathcal{P}$
4: *mark* c_i with *dfsnumber*(c_i)
5: *traverse_possible*(c_i)
6: start with initial data structure $\mathcal{D}$
7: **while** $\exists c_i, c_j \in \mathcal{P}$ s.t. $c_i \cap c_j \neq \emptyset$
8: **w.l.o.g. let** *dfsnumber*(c_i) <*dfsnumber*(c_j)
9: *delete*$(\mathcal{D}, C_i)$
10: *delete* c_i from $\mathcal{P}$, *add* $\overline{c_i}$ to $\mathcal{C}$
11: *traverse_certain*$(\overline{c_i})$
12: **if** there are intersecting circular arcs in $\mathcal{C}$
13: output **No**
14: **else**
15: output $\mathcal{C} \cup \mathcal{P}$

traverse_possible(c_i)
while C_j=*find*$(\mathcal{D}, c_i)$ not empty
delete$(\mathcal{D}, C_j)$
w.l.o.g. let c_j be involved in the intersection
(Ties are broken arbitrarily if both c_j and $\overline{c_j}$ are involved in the intersection.)
delete c_j from $\mathcal{P}$
mark $\overline{c_j}$ with *dfsnumber*$(\overline{c_j})$
traverse_possible$(\overline{c_j})$

traverse_certain(c_i)
while C_j=*find*$(\mathcal{D}, c_i)$ not empty
delete$(\mathcal{D}, C_j)$
w.l.o.g. let c_j be involved in the intersection
(Ties are broken arbitrarily if both c_j and $\overline{c_j}$ are involved in the intersection.)
delete c_j (or $\overline{c_j}$) from $\mathcal{P}$, *add* $\overline{c_j}$ to $\mathcal{C}$
traverse_certain$(\overline{c_j})$

Figure 3: Algorithm 2CAD. The input to the algorithm is $\mathcal{D}$, the data structure used to store all the circles. If it is possible to draw the graph without crossings, the algorithm outputs the set of circular arcs used to draw the graph.

There are two possibilities for each such intersection: either c_i intersects both c_j and $\overline{c_j}$, or c_i is an ancestor of c_j in the DFS tree. In the first case, clearly, $\overline{c_i} \in \mathcal{C}$. In the latter case, as $c_i \in \mathcal{C}$ implies $c_j \in \mathcal{C}$, we again have $\overline{c_i} \in \mathcal{C}$.

Initially $\mathcal{C}$ is empty. Once we find a circular arc, $\overline{c_i}$, that is certainly in the final solution, by the above observation, we perform a traversal from $\overline{c_i}$ (line 11) placing all certain arcs in the set $\mathcal{C}$. To find the intersections in $\mathcal{P}$ it suffices to perform a plane sweep over the set $\mathcal{P}$, which is incorporated into the while loop of line 7. At the beginning of the while loop of line 7 we have n arcs in $\mathcal{P}$, one arc for each circle C_i. As whenever we encounter an intersection in $\mathcal{P}$ we delete a circular arc, the plane sweep over $\mathcal{P}$ finds $O(n)$ intersections in total. Therefore, we first divide each circular arc $c_i \in \mathcal{P}$ into two subsegments c_i' and c_i'' such that each subsegment is *y-monotone*. We place the endpoints of all the subsegments into a balanced binary search tree $\mathcal{Q}$ which is the event queue of the plane sweep. We also store another balanced binary search tree $\mathcal{T}$ for the ordered sequence of segments intersecting the sweep line.

Unlike in a traditional plane sweep, whenever we encounter an intersection we delete a subset of the circular arcs from $\mathcal{P}$, in line 10 and line 11 as a result of *traverse_certain*. These deletions must be reflected in the plane sweep data structures $\mathcal{Q}$ and $\mathcal{T}$. As a result a simple plane sweep that runs in time $O((n+k)\log n)$, where k is the number of detected intersections, can be used to implement the intersection detection of the *while* loop of line 7. As the plane sweep encounters $O(n)$ intersections in total, we have $k = O(n)$. The *while* loop of line 7 requires $O(n \log n)$ time for the plane sweep and $O(n \times (\alpha(n) + \beta(n)))$ time for $O(n)$ *find/delete* operations, for a total of $O(n \log n + n \times (\alpha(n) + \beta(n)))$ time. Finally, we end up with a set $\mathcal{C}$ of certain circular arcs and a set $\mathcal{P}$ of possible circular arcs. Note that for every circle C_i exactly one of its arcs (c_i or $\overline{c_i}$) is either in $\mathcal{P}$ or $\mathcal{C}$ as the while loop of line 7 places one of the arcs in $\mathcal{C}$ while deleting one from $\mathcal{P}$. At this point we make a second observation:

Observation 2 *Let $c_i, c_j \in \mathcal{C} \cup \mathcal{P}$. At the end of the while loop of line 7, if $c_i \cap c_j \neq \emptyset$, then $c_i, c_j \in \mathcal{C}$.*

The observation holds for the following reasons. Assume $c_i, c_j \in \mathcal{P}$ were true. Then we must have encountered the intersection in the plane sweep step in which case one of them would have been deleted from $\mathcal{P}$. So, $c_i, c_j \in \mathcal{P}$ cannot be true. On the other hand, assume $c_i \in \mathcal{C}$ and $c_j \in \mathcal{P}$ were true. Then we must have encountered C_j before visiting c_i in the *traverse_certain* step. However, after we traverse through a circle, neither arc of the circle remains in $\mathcal{P}$. So this cannot be the case either.

Thus, we need to concentrate on the intersections in $\mathcal{C}$. We perform a final plane sweep over the set $\mathcal{C}$ in line 12. The plane sweep described above can be used with $\mathcal{C}$ as the input set of circular segments. The first intersection, if it exists, can be found in time $O(n \log n)$. If we encounter an intersection, then there cannot be an assignment without intersections, as that intersection would be between two certain circular arcs; otherwise, $\mathcal{C} \cup \mathcal{P}$ gives us a feasible assignment as the union contains exactly one arc for each circle C_i and there

exist no intersections between arc pairs in the union. The running time of the algorithm is the time required for the two *while* loops in the main algorithm: $O(n \log n + n \times (\alpha(n) + \beta(n)))$. In the next section we describe the data structure that supports the needed operations.

Note that Algorithm 2CAD can also be applied to solve the *Manhattan wiring problem* with the running time of $O(n \log n)$ which is the running time of the algorithm by Imai *et al.* [16]. We can design a data structure for a special kind of one-dimensional range search such that given a set of horizontal/vertical segments a sequence of $O(n)$ *LIST1/DELETE*'s can be executed in $O(n \log n)$ time, where *LIST1(I)* returns a segment intersecting an input segment I and *DELETE(I)* deletes the input segment I from the data structure [16]. We can modify the data structure so that each segment has a pointer to the rectangle it belongs to. Then each *find* operation to find a rectangle intersecting an input semi-rectangle consists of at most two *LIST1* operations on the data structure and each *delete* operation to delete a rectangle from the data structure consists of four *DELETE* operations.

3.2 The Data Structure

Given a circular arc query c_i, finding and deleting a circle C_j that intersects c_i is more efficient than performing the same operations on a circular arc c_j that intersects c_i. This observation led us to the 2CAD algorithm which assumes the existence of a data structure $\mathcal{D}$ that stores all the circles and allows for efficient *find/delete* operations. Gupta *et al.* [15] show how to reduce the problem of querying circles with a circular arc to half-space range searching in higher dimensions. The method requires at most a 4-dimensional half-space range searching. To report such intersections, we make use of the ideas from *geometric range-searching* [1, 2, 20]. The main data structure we use is a partition tree, constructed using the partitioning theorem by Matoušek [19]: a point set P can be partitioned into $O(n^{1-1/d})$ classes in time $O(n \log n^{1-1/d})$ such that for any class P_i, $|P_i| < 2n^{1/d}$ and any line intersects at most $O(n^{(1-1/d)^2})$ classes, where d is the dimension of the search space. In our case $d = 4$. Using this partitioning theorem we can create a data structure $\mathcal{D}'$ that performs half-space range queries in time $O(n^{1-1/d}(\log n)^{O(1)})$. Moreover, $\mathcal{D}'$ is dynamic, in the sense that we can delete a circle from $\mathcal{D}'$ in amortized time $O(\log n)$. Then using multiple levels of $\mathcal{D}'$ to satisfy the intersection conditions of [15] we create the data structure $\mathcal{D}$ that supports $find(\mathcal{D}, c_i)$ operations in $O(n^{3/4} polylog\ n)$ time and that requires $O(n \log n)$ time for a sequence of $O(n)$ $delete(\mathcal{D}, C_i)$ operations. These results can be summarized with the following theorem for the 2CAD problem.

Theorem 1 *The 2CAD problem can be solved in time* $O(n^{7/4} polylog\ n)$.

3.3 Allequal2CAD

We can solve a restricted version of the 2CAD problem even more efficiently. Let *Allequal2CAD* be the version of 2CAD where each circle C_i has the same

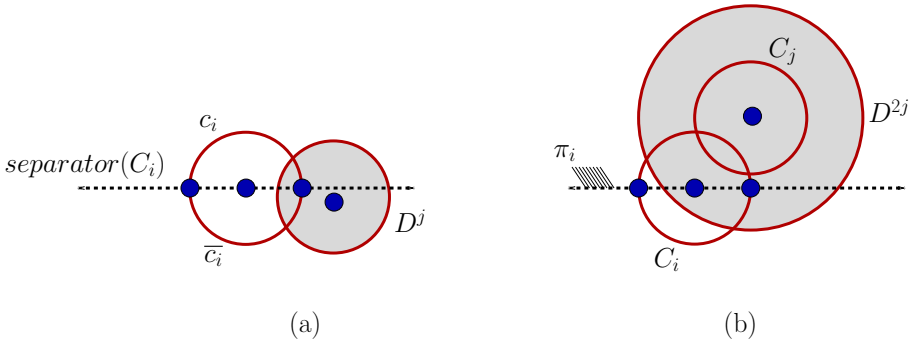


Figure 4: (a) D^j contains an endpoint of c_i (b) center$(C_j) \in \pi_i$ and center$(C_i) \in D^{2j}$.

diameter and for every edge e_i the line segment between the endpoints of e_i is the diameter of circle C_i. In this case, for a given edge e_i, the circular arcs c_i and $\overline{c_i}$ are half-circles. We present an algorithm to solve Allequal2CAD in $O(n^{3/2} polylog\, n)$ time using a data structure $\mathcal{D}$ that enables us to perform efficient *find/delete* operations. We provide the details for the construction of $\mathcal{D}$ here as the general data structure described above can be constructed in a similar fashion.

Let $center(C_i)$ and $separator(C_i)$ denote, respectively, the center of C_i and the line separating the half-circles c_i and $\overline{c_i}$. Define π_i as the half-plane bounded by separator(C_i) that contains c_i. Let D^i be the disk bounded by the circle C_i, and let D^{2i} be the disk concentric to D^i but with radius twice the radius of D^i; see Fig 4. The following lemma is easy to verify.

Lemma 1 *A circle C_j intersects a half-circle c_i if and only if* **(i)** *D^j contains an endpoint of c_i, or* **(ii)** *$center(C_j) \in \pi_i$ and $center(C_i) \in D^{2j}$.*

In order to report intersections of the first type we use the data structure described by Efrat *et al.* [10]: given n equal-sized disks in the plane, construct a data structure $\mathcal{DT}_1$ in time $O(n \log n)$ such that for a given query point p, finding a disk that contains p requires $O(\log n)$ time. Moreover, deleting a disk from $\mathcal{DT}_1$ requires amortized time $O(\log n)$. We preprocess the set of disks $D^1 \ldots D^n$ by setting up this structure.

To deal with the intersections of the second type we make use of the partition tree $\mathcal{D}'$ described above. However, this time we perform half-space range searching in two dimensions using a two-level data structure. Let the data structure for the second type of intersections be $\mathcal{DT}_2$. The first level of $\mathcal{DT}_2$ is a partition tree, $\mathcal{DT}_2{}'$. Based on the partitioning theorem described above, we partition the centers of all the circles and recursively build the partition tree $\mathcal{DT}_2{}'$. The leaves of $\mathcal{DT}_2{}'$ partition the centers into constant-sized subsets. Each internal node v is associated with a subset P_v of the points contained in the leaves of the subtree rooted at v. We build the second level of the data structure based on these subsets. The second level data structure used in $\mathcal{DT}_2$ is the same as $\mathcal{DT}_1$,

except we preprocess the disks D^{2i} for each i rather than the disks D^i as is the case for constructing $\mathcal{DT}_1$. We call this second level data structure $\mathcal{DT}_2''$ to distinguish it from $\mathcal{DT}_1$ which we used to find the first type of intersections. Each internal node v in $\mathcal{DT}_2'$ contains a pointer to the corresponding $\mathcal{DT}_2''$, where $\mathcal{DT}_2''$ contains the data structure for all the disks D^{2i} centered at P_v. The preprocessing time for constructing the partition in a node of $\mathcal{DT}_2'$ with m points is $O(m \log m)$. Constructing $\mathcal{DT}_2''$ for the same node also takes time $O(m \log m)$. As the number of points in nodes of $\mathcal{DT}_2'$ decreases as a double exponential with their depth in the tree, the total preprocessing time is $O(n \log n)$.

Theorem 2 *Allequal2CAD problem can be solved in time* $O(n^{3/2} polylog\ n)$.

Proof: To find a circle C_j intersecting a given half-circle c_i we first query $\mathcal{DT}_1$ with c_i's endpoints. This step requires $O(\log n)$ time. If we cannot find such a circle then we query $\mathcal{DT}_2$ with separator(C_i). Upon finding an internal node v such that P_v lies completely above separator(C_i), we query the associated $\mathcal{DT}_2''$ of v with center(C_i). Let $\alpha(n)$ be the time to find a circle intersecting a given half-circle c_i. Then $\alpha(n)$ is bounded by the query time of $\mathcal{DT}_2$ and we get:

$$\alpha(n) \leq O(\sqrt{n}) \times \log 2\sqrt{n} + O(\sqrt[4]{n}) \times \alpha(2\sqrt{n}) \tag{1}$$

Thus, the time required to perform a *find* operation is $\alpha(n) = O(\sqrt{n} polylog\ n)$.

In order to delete a circle C_i we first delete D^i from $\mathcal{DT}_1$ in $O(\log n)$ amortized time. We also need to delete the appropriate disks in $\mathcal{DT}_2$. To do this we simply find each internal node v of $\mathcal{DT}_2'$ such that center$(C_i) \in P_v$ and delete the corresponding disk from $\mathcal{DT}_2''$, the second level data structure pointed to by v. As $\mathcal{DT}_2'$ has depth $O(\log \log n)$ and deleting a disk from $\mathcal{DT}_2''$ takes amortized time $O(\log n)$, the deletion of a circle takes $O(\log^2 n)$ amortized time. As *find* and *delete* operations are defined for both $\mathcal{DT}_1$ and $\mathcal{DT}_2$, the two data structures form the complete data structure $\mathcal{D}$ and the theorem follows. □

4 The 3CAD Problem

The 3CAD problem is similar to 2CAD, except now we have three choices for the drawing of each edge $e_i = (u, v)$. We show that the 3CAD problem is NP-hard even for a restricted version of the problem, where for each edge e_i the three possible circular arcs are determined by the line segment between the endpoints of e_i: the two half circles of the circle C_i with the line segment as its diameter and the line segment itself constituting a circular arc of infinite radius. We show this restricted version of 3CAD is NP-hard using a reduction from the NP-hard PLANAR-3SAT [18].

A 3SAT instance Φ is called a PLANAR-3SAT instance if the (bipartite) occurrence graph $G_\Phi = (V_\Phi, E_\Phi)$ is planar. In the occurrence graph G_Φ the vertex set V_Φ contains a vertex for each variable and clause, and edge set E_Φ contains an edge between two vertices $v, w \in V_\Phi$ if v represents a variable x that

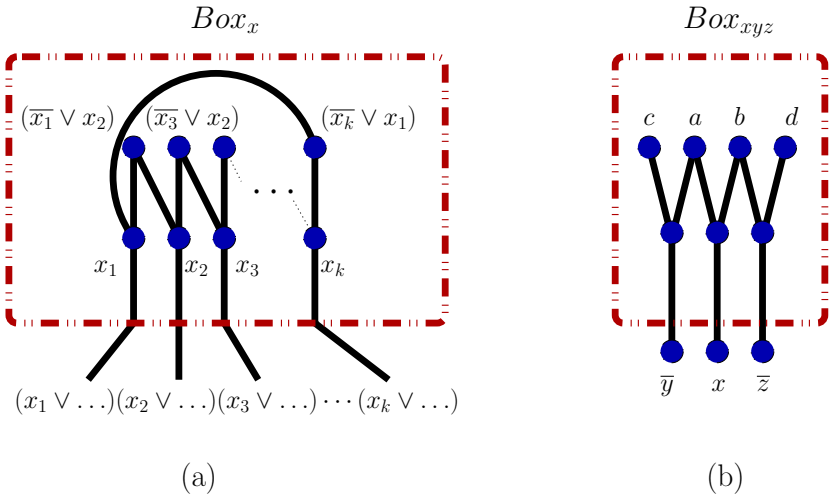


Figure 5: (a) VR gadget; (b) 1-IN-3 gadget.

occurs in the clause represented by w. Let VR3SAT(Variable Restricted 3SAT) be the version of 3SAT with the restriction that each variable can appear at most three times, and VR1IN3SAT be the version of VR3SAT in which *exactly* one literal in each clause is required to be true. In the planar versions of these two problems the occurrence graphs of the input instances must be planar. We will convert a PLANAR-3SAT instance Φ into a 3CAD instance Φ_{3CAD} through a series of modifications that preserve planarity.

Lemma 2 *PLANAR-VR3SAT is NP-hard.*

Proof: The reduction is from PLANAR-3SAT. Let Φ be an input formula for 3SAT. The traditional reduction from 3SAT to VR3SAT converts Φ into Φ' in such a way that if x is a variable appearing k times in Φ then in Φ' the first occurrence of x is replaced by x_1, the second by x_2, and so on, where x_1, x_2, …, x_k are k new variables [22]. Φ' also includes the new clauses $(\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee x_3) \wedge \ldots \wedge (\overline{x_k} \vee x_1)$. We apply the same reduction to convert a PLANAR3SAT instance Φ into a PLANAR-VR3SAT instance Φ'. We show that if Φ is planar then so is Φ'. Let G_Φ, $G_{\Phi'}$ be the occurrence graphs of Φ, Φ' respectively, and let v_x be the vertex corresponding to the variable x of Φ in G_Φ. For constructing $G_{\Phi'}$, we replace v_x with a box_x, where box_x contains the occurrence graph of the newly added clauses $(\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee x_3) \wedge \ldots \wedge (\overline{x_k} \vee x_1)$ and the newly added variables x_1, x_2, …, x_k. It is easy to see that the occurrence graph inside box_x is planar. The vertices corresponding to the clauses containing x can still be incident to the vertices inside box_x without affecting planarity, as shown in Fig. 5(a). This implies that $G_{\Phi'}$ is also planar. □

Lemma 3 *PLANAR-VR1IN3SAT is NP-hard.*

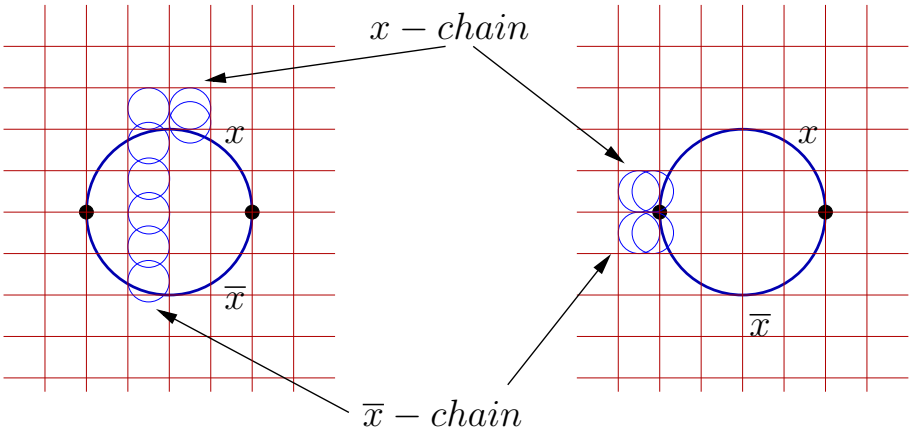


Figure 6: **Left:** Configuration of variable-circle and the connected 2-link chain going up. Note that only the lower half-circle of the first circle in the $x - chain$ intersects x, and only the upper half-circle of the fifth circle in the $\overline{x} - chain$ intersects x. Similarly only the lower half-circle of the first circle in the $\overline{x} - chain$ intersects $\overline{x}$. 2-link chain going down can be constructed similarly. **Right:** Configuration of variable-circle and the connected 2-link chain going left. 2-link chain going right can be constructed similarly.

Proof: The reduction from VR3SAT to VR1IN3SAT converts Φ', an instance of VR3SAT, to Φ'' so that each clause $(x \vee y \vee z)$ is replaced by the clauses $(x \vee a \vee b) \wedge (\overline{y} \vee a \vee c) \wedge (\overline{z} \vee b \vee d)$, and each clause $(x \vee y)$ is replaced by the clauses $(x \vee a) \wedge (\overline{y} \vee a \vee c)$. Let $G_{\Phi'}$ be the occurrence graph of Φ', and v_{xyz} be the vertex corresponding to the clause $(x \vee y \vee z)$ of Φ' in $G_{\Phi'}$. We replace v_{xyz} with a box_{xyz} in $G_{\Phi''}$, where box_{xyz} contains the occurrence graph of the newly added clauses $(x \vee a \vee b) \wedge (\overline{y} \vee a \vee c) \wedge (\overline{z} \vee b \vee d)$ and the newly added variables a, b, c, d. It follows that $G_{\Phi''}$ is also planar; see Fig. 5(b). □

Theorem 3 *The 3CAD problem is NP-hard.*

Proof: We convert a PLANAR-VR1IN3SAT instance Φ'' into a 3CAD instance Φ_{3CAD}. Note that the occurrence graph $G_{\Phi''}$ for Φ'' has maximum degree 3 because of the VR constraint. Then there exists an orthogonal drawing for $G_{\Phi''}$ (a drawing such that each vertex is on the integer grid and each edge consists of horizontal and vertical edge segments). An orthogonal drawing of $G_{\Phi''}$ where the grid is of size quadratic in the size of $G_{\Phi''}$ can be found in linear time [25]. Given the orthogonal drawing of $G_{\Phi''}$, we obtain Φ_{3CAD} by the following method. We begin by enlarging the grid 16 times. Note that the size of the new grid is still quadratic in the size of $G_{\Phi''}$. We replace each vertex corresponding to a variable x with a *variable-circle*, with one half labeled x and the other $\overline{x}$. Each variable-circle has diameter equal to four times the width of a grid square. We replace each vertex corresponding to a clause, say, $(x \vee y \vee z)$ of Φ'', with a *clause-circle* having one half-circle labeled x, one labeled y, and

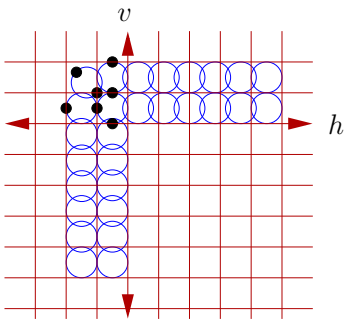


Figure 7: 2-link chain making a right turn. Half-circles of the chain-circles below line h are separated by a horizontal diameter, half circles of the ones to the right of line v are separated by a vertical diameter. The remaining endpoints are as shown.

the diameter of the circle labeled z. and each clause-circle has diameter equal to eight times the grid square width.

We represent each edge of $G_{\Phi''}$ with a *2-link-chain* which consists of two parallel links of *chain-circles*, each with diameter equal to the width of a grid square. Let e be the edge of $G_{\Phi''}$ between the vertex corresponding to the variable x and the vertex corresponding to the clause $(x \vee y \vee z)$. Then we represent e by a 2-link-chain, where one of the links, $x - chain$, is connected to x and is used for the flow of the truth assignment to x. The second link-chain, $\overline{x} - chain$, is used for the flow of the assignment to $\overline{x}$ and is connected to $\overline{x}$ in the variable-circle; see Fig. 6. We assume that the direction of the flow is from a variable-circle to a clause-circle. Except for the edge-turns and the connections to a clause-circle, the half-circles of a chain-circle are determined by the drawing of the edge in the orthogonal drawing of $G_{\Phi''}$; see Fig. 7. If the edge segment containing the chain-circle is drawn vertically, then the half-circles are separated by a horizontal diameter; otherwise, the half-circles are separated by a vertical diameter.

In order to guarantee a correct flow of truth assignments to the clauses we connect the link-chains to the clause-circles as shown in Fig. 8. At the top we show the case where x is assigned to the top-half circle of the clause-circle. If the link-chain connects from below, then the connections are similar to those in the left figure (connections from above). If the link-chain connects from left, then they are similar to those in the right figure (connections from right). At the bottom we show the case where x is assigned to the diagonal of the clause-circle. Again we can use similar connections if the connections are from below or from left. The cases where x is assigned to the bottom half-circle of the clause-circle are symmetrical to those of the top figures, where x is assigned to the top half-circle.

The number of variable/clause circles introduced is linear in terms of the size of $G_{\Phi''}$. The number of chain-circles introduced is quadratic in the size of $G_{\Phi''}$ as every seven adjacent chain-circles are completely embedded within six

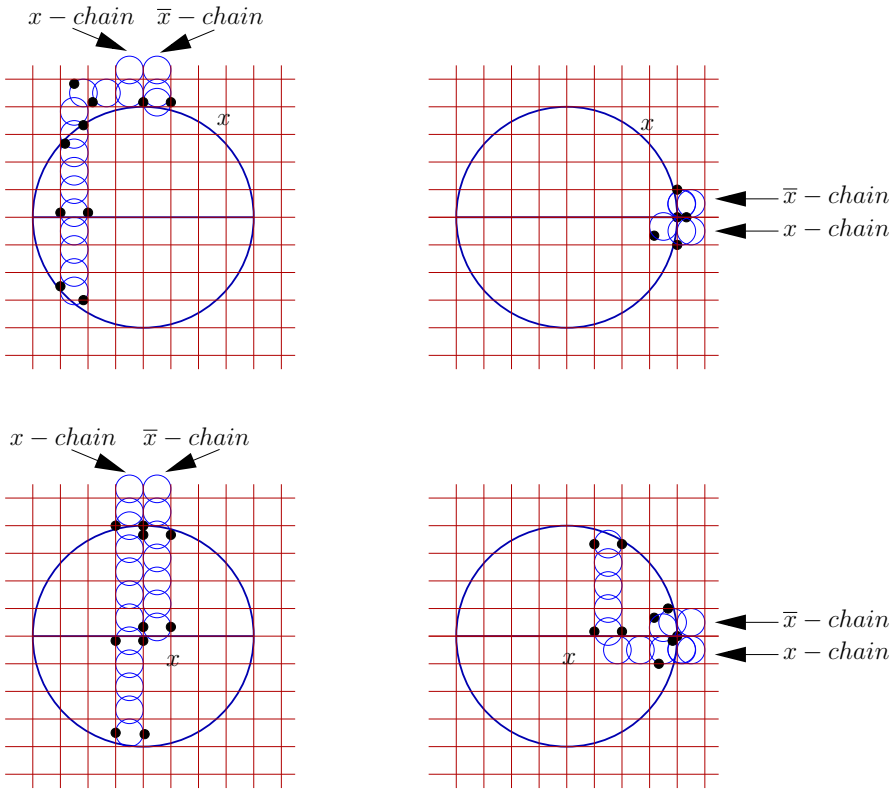


Figure 8: Clause-circle connections. **Top:** x is assigned to the top half-circle of clause-circle. **Left:** Connecting 2-link chain carrying flow from variable-circle x from above. **Right:** Connecting 2-link chain carrying flow from variable-circle x from right. **Bottom:** x is assigned to the diagonal of clause-circle. **Left:** Connecting 2-link chain carrying flow from variable-circle x from above. **Right:** Connecting 2-link chain carrying flow from variable-circle x from right.

grid squares. Thus, the whole construction can be completed in quadratic time with regard to the size of $G_{\Phi''}$.

We claim that Φ'' is satisfiable if and only if Φ_{3CAD} has a feasible assignment without crossings. Assume that Φ'' is a satisfiable instance of PLANAR-VR1IN3SAT, and let α be a satisfying assignment. A feasible assignment of edges in Φ_{3CAD} can be obtained as follows. For each variable-circle corresponding to variable x, assign the half-circle labeled with x or $\overline{x}$ depending on whether x is assigned to true or false in α respectively. For each clause-circle corresponding to a clause $(x \vee y \vee z)$, assign the half-circle (or diameter) corresponding to the (only) true literal in the clause, as determined by α. For each 2-link-chain connected to the variable-circle of x, if x is assigned to true in α, then for the link that is connected to x, assign the first chain-circle by choosing the half-circle that *does not* cross the x, and continue assigning the chain-circles through the

link without creating any crossings. For the second link that is connected to the $\overline{x}$ half-circle, assign the first chain-circle by choosing the half-circle that *crosses* the half-circle $\overline{x}$, and continue assigning the chain-circles through the link without creating any crossings. This assignment does not contain any crossings. The only crossings that could occur would be between a chain-circle at the tip of a link and a clause-circle, but our method of assigning the chain-circles eliminates this possibility.

For the other direction, assume that Φ_{3CAD} has a feasible assignment of edges without crossings. Then, finding a truth assignment α for Φ'' is straightforward. For each variable-circle corresponding to a variable x, if the half-circle labeled with x is chosen, then assign x true; otherwise, assign it false. This yields a satisfying assignment, as the feasible assignment of edges in Φ_{3CAD} chooses exactly one edge from each clause-circle such that there are no conflicts with the variable assignments and the true literal assignment for the other clauses. □

5 Drawing with Few Crossings

If G cannot be drawn without crossings using two circular arcs, there are two natural optimization problems that can be defined as decision problems. Define Min2CAD as the following decision problem: given $(G = (V, E), \kappa_{min})$, where G is a planar graph, and κ_{min} is a non-negative integer, does there exist an assignment of circular arcs (either c_i or $\overline{c_i}$) for *each* $e_i \in E$ such that the number of crossings is at most κ_{min}? The second decision problem, Max2CAD, is defined as follows: given $(G = (V, E), \kappa_{max})$, where G is a planar graph, and κ_{max} is a non-negative integer, does there exist an assignment of circular arcs (either c_i or $\overline{c_i}$) for *some* $e_i \in E$ such that there are no crossings and the number of assigned edges is at least κ_{max}? We prove that both problems are NP-hard by reductions from the Planar Degree-4 Independent Set problem (PD4IS). Let $H = (V_H, E_H)$ be an undirected graph. We say that a set $I \subseteq V_H$ is independent if for all pairs (i, j), where $i, j \in I$, $(i, j) \notin E_H$. The PD4IS problem is the following: given $(H = (V_H, E_H), \kappa_{IND})$, where H is a planar graph with maximum degree 4 and κ_{IND} is a non-negative integer, does there exist an independent set I with |I|=κ_{IND}?

Lemma 4 *PD4IS is NP-hard.*

Proof: The proof is based on a reduction from PLANAR-VR1IN3SAT. Let ϕ be an instance of PLANAR-VR1IN3SAT with p clauses $C_1, C_2, \ldots, C_p$, where $C_i = (l_{i1} \vee l_{i2} \vee l_{i3})$ or $C_i = (l_{i1} \vee l_{i2})$. Each l_{ij} is either x_k or $\overline{x_k}$, where x_k is one of the q variables. The reduction constructs an instance of PD4IS, $(H = (V_H, E_H), \kappa_{IND})$ as follows: $V_H = \{l_{ij} : i = 1, \ldots, p; j = 1, 2, 3\} \cup \{x_k : k = 1, \ldots, q\} \cup \{\overline{x_k} : k = 1, \ldots, q\}$ and $E_H = \{(l_{ij}, l_{ik}) : i = 1, \ldots, p; j \neq k\} \cup \{(l_{ij}, x_k)$ if $l_{ij} = \overline{x_k}$ or $(l_{ij}, \overline{x_k})$ if $l_{ij} = x_k : i = 1, \ldots, p; j = 1, 2, 3; k = 1, \ldots, q\} \cup \{(x_k, \overline{x_k}) : k = 1, \ldots, q\}$ and $\kappa_{IND} = p + q$; see Fig. 9. Note that because we start with an instance of planar 3SAT, graph H is also planar.

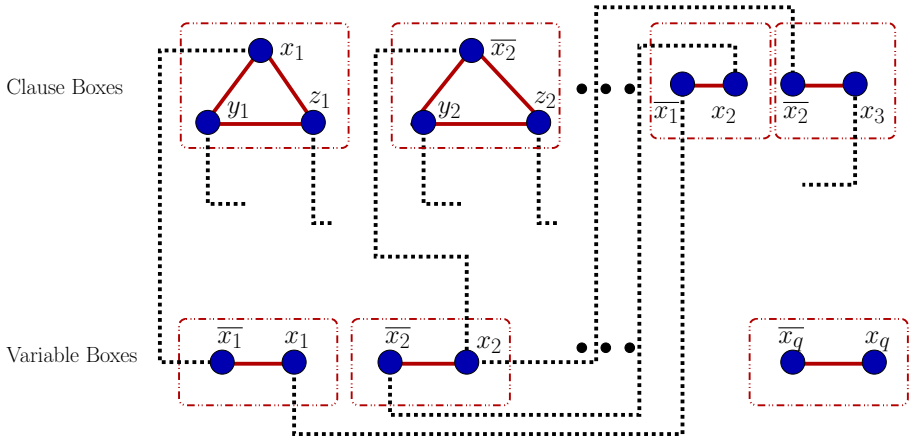


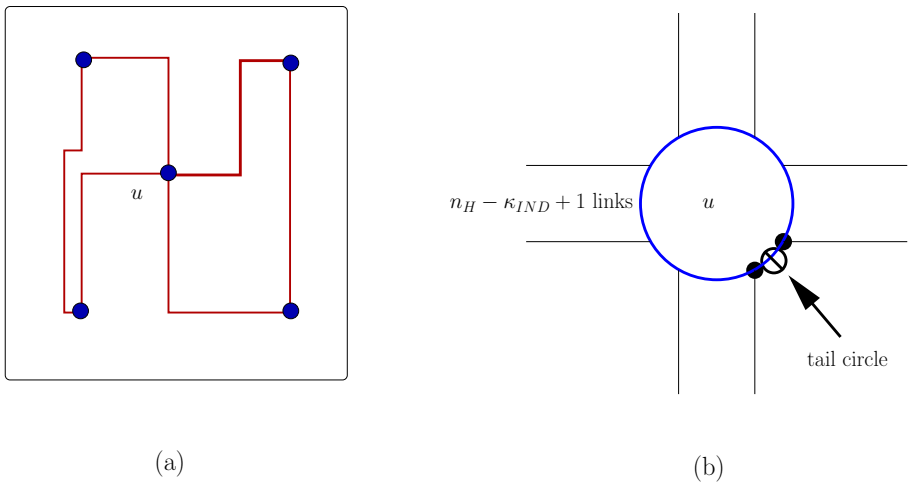
Figure 9: The PD4IS graph H to which the PLANAR-VR1IN3SAT instance $\phi = (x_1 \vee y_1 \vee z_1) \wedge (\overline{x_2} \vee y_2 \vee z_2) \wedge \ldots \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee x_1) \wedge \ldots$ is reduced.

Moreover, because the instance ϕ also has the property that each variable occurs at most three times, H will have degree 4.

There exists an independent set I of H with size $\kappa_{IND} = p+q$ if and only if ϕ is satisfiable. Assume there exists an independent set I of size $p+q$. I must contain a node from each box because each box contains a clique and there are $p+q$ boxes; see Fig. 9. As the nodes are labeled with literals, and I does not contain nodes corresponding to opposite literals, I corresponds to a truth assignment that satisfies ϕ. Now we assume that a satisfying assignment for ϕ exists. Then we can identify a true literal in each clause c_i. Let a true literal in c_i be l_i. Then we can pick the vertex corresponding to l_i from the clause box for c_i, and the vertex corresponding to the literal l_i from the variable box for x_i in H to be in I. This yields $p+q$ independent vertices in H. □

Theorem 4 *Min2CAD is NP-hard.*

Proof: Let $(H = (V_H, E_H), \kappa_{IND})$ be an instance of PD4IS. Let n_H denote the size of V_H. The reduction produces a Min2CAD instance (G, κ_{min}), with $\kappa_{min} = n_H - \kappa_{IND}$. As H has maximum degree 4, we can find an orthogonal drawing of H, such that each vertex is on the integer grid of size quadratic in the size of H [25]; see Fig. 10(a). The reduction scales the grid of H by a factor of $2 \times (n_H - \kappa_{IND} + 1)$ and replaces the vertices of H with *vertex-circles*, circles of diameter $2 \times (n_H - \kappa_{IND} + 1)$ units. We place two endpoints in the fourth quadrant. One endpoint is placed so that it lies on the circle and has $x-$distance $(n_H - \kappa_{IND} + 1)/2$ from the center of the circle and the other endpoint at $y-$distance $(n_H - \kappa_{IND} + 1)/2$ from the center; see Fig. 10(b). These two endpoints define two circular arcs. We call the larger arc the *head* and the smaller one the *tail*. Each edge of H is represented with $n_H - \kappa_{IND} + 1$ links of *chain-circles*, circles having half a unit diameter connected to a vertex-circle



(a) (b)

Figure 10: (a) The original graph H drawn on an integer grid. (b) Min2CAD reduction shown for vertex u. Each *vertex-circle* (the big circle) corresponds to a vertex in H, *chain-circles* lie inside rectangular pipes connecting to a vertex-circle and correspond to edges in H, and *tail-circles* are auxiliary circles. There are $n_H - \kappa_{IND} + 1$ links of chain circles in each pipe. Each chain circle has endpoints lying on the horizontal diameter and contains two half-circles.

at its *head*. Each vertex-circle also has a *tail-circle*, connected to it at its *tail*, in such a way that the diameter of the tail-circle crosses the tail of the vertex-circle. As the given graph H is planar, we can obtain such a grid drawing of circles without causing any intersection between the chain-circles. The resulting drawing induces graph G, where each endpoint of a circle corresponds to a vertex and each circle itself corresponds to an edge between the vertices representing the endpoints.

We claim that H has an independent set of size κ_{IND} if and only if G can be drawn using circular arcs with at most $n_H - \kappa_{IND}$ crossings. Assume H has an independent set I, where $|I|{=}\kappa_{IND}$. Then there are κ_{IND} vertices in H that are pairwise disconnected, which further implies that in G there are κ_{IND} vertex-circles which are not connected to each other by links. Then a feasible assignment of half-circles which allows a drawing of G with at most $n_H - \kappa_{IND}$ crossings follows easily. For each vertex-circle, if the vertex corresponding to it is in I, then assign the head of the vertex-circle; otherwise, assign the tail as chosen. This results in an assignment that will have κ_{IND} heads and $n_H - \kappa_{IND}$ tails. The chain-circles of the links connected to a vertex-circle which is already assigned to its head are assigned so that no crossing is created, i.e., starting from the chain-circle attached to the already assigned head, choose the half-circle that does not create any crossings.

The chain-circles of the other links are assigned edges in a similar fashion, but this time without the condition on the assignment of the first chain-circle. Finally the tail-circles are assigned arbitrarily to the half-circles. Such an as-

signment assigns circular arcs for every circle in the drawing and creates no more than $n_H - \kappa_{IND}$ crossings. The only crossings created are those between the tail-circles and the vertex-circles assigned to their tails. We already know that there are $n_H - \kappa_{IND}$ such vertex tails, which implies the first direction of the claim.

For the other direction, assume G has an assignment of circular arcs with at most $n_H - \kappa_{IND}$ crossings. Let C_{head} (C_{tail}) be the sets of vertex-circles having their heads (respectively tails) chosen in this assignment. As the assignment creates no more than $n_H - \kappa_{IND}$ crossings, we have $\mid C_{\text{tail}} \mid \leq n_H - \kappa_{IND}$. This implies that $\mid C_{\text{head}} \mid \geq \kappa_{IND}$, as $\mid C_{\text{tail}} \mid + \mid C_{\text{head}} \mid = n_H$. For any pair (c_i, c_j), where c_i, $c_j \in C_{\text{head}}$, there cannot be any links between c_i and c_j because if c_i and c_j were linked together, each of the $n_H - \kappa_{IND} + 1$ links would have at least one crossing, creating more than $n_H - \kappa_{IND}$ crossings which would be a contradiction. Let $I \subseteq V_H$ be the set of vertices corresponding to the vertex-circles in C_{head}; then I is an independent set of size κ_{IND}. □

Theorem 5 *Max2CAD is NP-hard.*

Proof: The reduction is again from PD4IS. Let $(H = (V_H, E_H), \kappa_{IND})$ be an instance of PD4IS. The reduction produces a Max2CAD instance (G, κ_{max}), with $\kappa_{max} = t_H - n_H + \kappa_{IND}$, where t_H is the total number of circles in G. The proof proceeds along the lines of the proof of Theorem 4 with a slight modification: in this case we add $n_H - \kappa_{IND} + 1$ tail-circles to each vertex-circle, rather than just one tail-circle. Then H has an independent set of size κ_{IND} if and only if G can be drawn without any crossings such that at least $\kappa_{max} = t_H - n_H + \kappa_{IND}$ edges have been assigned to some circular arc. □

6 Conclusion and Open Problems

We presented two algorithms for the 2CAD problem and showed that two natural optimization versions are NP-hard. We also showed that the 3CAD problem is NP-hard. We conclude with several related open problems:

1. Can Min2CAD be approximated within a constant factor?
2. Can Max2CAD be approximated within a constant factor?
3. Can we draw graphs with pre-specified vertex positions, using elliptic or other parabolic curve segments without creating too many crossings?
4. A closely related open problem is that of partial embeddability. Brandenberg *et.al.* [5] posed the following problem: given a planar graph in which some vertices have already been placed in the plane (i.e., given a partial embedding) is there a polynomial time algorithm that places the remaining vertices such that no two (straight-line) edges intersect? What if we are allowed to use circular arcs?

7 Acknowledgments

We would like to thank Roberto Tamassia, Sue Whitesides, and Helmut Alt for the stimulating discussions and suggestions.

References

[1] P. Agarwal. Range searching. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, CRC Press*, pages 575–598. 1997.

[2] P. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. Goodman, and R. Pollack, editors, *Advances in Discrete and Comput. Geom., American Mathematical Society, Providence*. 1998.

[3] P. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Computational Geometry: Theory and Applications*, 11:209–218, 1998.

[4] B. Apswall, M. Plass, and R. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Proc. Letters*, 8:121–123, 1979.

[5] F.-J. Brandenburg, D. Eppstein, M. T. Goodrich, S. G. Kobourov, G. Liotta, and P. Mutzel. Selected open problems in graph drawing. In *Proceedings on Graph Drawing (GD)*, pages 515–539, 2003.

[6] C. C. Cheng, C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Drawing planar graphs with circular arcs. *Discrete and Computational Geometry*, 25:405–418, 2001.

[7] S. Doddi, M. Marathe, A. Mirzaian, B. Moret, and B. Zhu. Map labeling and its generalizations. In *8th Symposium on Discrete Algorithms*, pages 148–157, 1997.

[8] S. Doddi, M. Marathe, and B. Moret. Point set labeling with specified positions. *Int. J. Computational Geometry & Applications*, 12(1-2):29–66, 2002.

[9] P. Eades and K. Sugiyama. How to draw a directed graph. *J. Inform. Process.*, 13:424–437, 1991.

[10] A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31:1–28, 2001.

[11] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5:691–703, 1976.

[12] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 281–288, 1991.

[13] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Softw. – Pract. Exp.*, 21(11):1129–1164, 1991.

[14] M. Godau. On the difficulty of embedding planar graphs with inaccuracies. In *Proceedings on Graph Drawing (GD'94)*, pages 254–261, 1994.

[15] P. Gupta, R. Janardan, and M. Smid. Algorithms for some intersection searching problems involving circular objects. *Intl. J. of Math. Algorithms*, 1:35–52, 1999.

[16] H. Imai and T. Asano. Efficient algorithms for geometric graph search problems. *SIAM J. Comput.*, 15:478–494, 1986.

[17] M. Kaufmann and R. Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *Journal of Graph Algorithms and Applications*, 6(1):115–129, 2002.

[18] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11:329–343, 1982.

[19] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.

[20] J. Matoušek. Geometric range searching. *ACM Computing Surveys*, 26:421–462, 1994.

[21] J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. In *Proceedings on Graph Drawing (GD'98)*, pages 263–274, 1998.

[22] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, 1994.

[23] C. Poon, B. Zhu, and F. Chin. A polynomial time solution for labeling a rectilinear map. *Information Processing Letters*, 65(4):201–207, 1998.

[24] T. Strijk and M. van Kreveld. Labeling a rectilinear map more efficiently. *Information Processing Letters*, 69(1):25–30, 1999.

[25] R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Trans. Circuits Syst.*, CAS-36(9):1230–1234, 1989.