# UC Irvine
## UC Irvine Previously Published Works

**Title**
Streamed Graph Drawing and the File Maintenance Problem

**Permalink**
https://escholarship.org/uc/item/3jb503n3

**Authors**
Goodrich, Michael T
Pszona, Paweł

**Publication Date**
2013

**DOI**
10.1007/978-3-319-03841-4_23

Peer reviewed

# Streamed Graph Drawing and the File Maintenance Problem

Michael T. Goodrich and Paweł Pszona

Dept. of Computer Science, University of California, Irvine

**Abstract.** In *streamed graph drawing*, a planar graph, $G$, is given incrementally as a data stream and a straight-line drawing of $G$ must be updated after each new edge is released. To preserve the mental map, changes to the drawing should be minimized after each update, and Binucci *et al.* show that exponential area is necessary and sufficient for a number of streamed graph drawings for trees if edges are not allowed to move at all. We show that a number of streamed graph drawings can, in fact, be done with polynomial area, including planar streamed graph drawings of trees, tree-maps, and outerplanar graphs, if we allow for a small number of coordinate movements after each update. Our algorithms involve an interesting connection to a classic algorithmic problem—the *file maintenance problem*—and we also give new algorithms for this problem in a framework where bulk memory moves are allowed.

## 1  Introduction

In the *streamed graph drawing* framework, which was introduced by Binucci *et al.* [4,3], a graph, $G$, is incrementally presented as a data stream of its vertices and edges, and a drawing of $G$ must be updated after each new edge is released. So as to preserve the *mental map* [6,8] of the drawing, this framework also requires that changes to the drawing of $G$ should be minimized after each update. Indeed, to achieve this goal, Binucci *et al.* took the extreme position of requiring that once an edge is drawn no changes can be made to that edge. They showed that, under this restriction, exponential area is necessary and sufficient for planar drawings of trees under various orderings for how the vertices and edges of the trees are presented.

In light of recent results regarding the mental map [1], however, we now know that moving a small number of vertices or edges in a drawing of a graph does not significantly affect readability in a negative way. Therefore, in this paper, we choose to relax the requirement that there are no changes to the drawing of the graph after an update and instead allow a small number of coordinate movements after each such update. In this

paper, we study planar streamed graph drawing schemes for trees, tree-maps, and outerplanar graphs, showing that polynomial area is achievable for such streamed graph drawings if small changes to the drawings are allowed after each update. Our results are based primarily on an interesting connection between streamed graph drawing and a classic algorithmic problem, the *file maintenance problem.*

In the *file maintenance problem* [11], we wish to maintain an ordered set, $S$, of $n$ elements, such that each element, $x$ in $S$, is assigned a unique integer label, $L(x)$, in the range $[0, N]$, where $x$ comes before $y$ if and only if $L(x) < L(y)$. In the classic version of this problem, $N$ is restricted to be $O(n)$, with the motivation that the integer labels are addresses or pseudo-addresses for memory locations where the elements of $S$ are stored[1]. If $N$ is only restricted to be polynomial in $n$, then this is known as the *online list labeling problem* [2,5]. In either case, the set, $S$, can be updated by issuing a command, insertAfter$(x, y)$, where $y$ is to be inserted to be immediately after $x \in S$ in the ordering, or insertBefore$(x, y)$, where $y$ is to be inserted to be immediately before $x \in S$ in the ordering. The goal is to minimize the number of elements in $S$ needing to be relabeled as a result of such an update.

***Previous Related Results.*** For the file maintenance problem, Willard [11] gave a rather complicated solution that achieves $O(\log^2 n)$ relabelings in the worst case after each insertion, and this result was later simplified by Bender *et al.* [2]. For the online list labeling problem, Dietz and Sleator [5] give an algorithm that achieves $O(\log n)$ amortized relabelings per insertion, and $O(\log^2 n)$ in the worst-case, using an $O(n^2)$ tag range. Their solution was simplified by Bender *et al.* [2] with the same bounds. Recently, Kopelowitz [7] has given an algorithm that achieves $O(\log n)$ worst-case relabelings after each insertion, using a polynomial bound for $N$.

For streamed tree drawings, as we mentioned above, Binucci *et al.* [4,3] show that exponential area is required for planar drawings of trees, depending on the order in which vertices and edges are introduced (e.g, BFS, DFS, etc.).

***Our Results.*** For the context of this paper, we focus on planar drawings of graphs, so we consider a drawing to consist essentially of a set of non-crossing line segments. For traditional drawings of trees and outerplanar

---

[1] For instance, in the EDT text editor developed for the DEC PDP-11 series of mini-computers, each line was assigned a pseudo line number, 1.0000, 2.0000, and so on, and if a new line was to be introduced between two existing lines, $x$ and $y$, it was given as a default label the average of the labels of $x$ and $y$ as its label.

graphs, the endpoints of the segments correspond to vertices and the segments represent edges. In tree-map drawings, each vertex $v$ of a tree $T$ is represented by a rectangle, $R$, such that the children of $v$ are represented by rectangles inside $R$ that share portions of at least two sides of $R$. Thus, in a tree-map drawing, the line segments correspond to the sides of rectangles.

We present new streamed graph drawing algorithms for general trees, tree-maps, and outerplanar graphs that keep the area of the drawing to be of polynomial size and allow new edges to arrive in any order, provided the graph is connected at all times. After each update to a graph is given, we allow a small number of, say, a polylogarithmic number of the endpoints of the segments in the drawing to move to accommodate the representation of the new edge. We alternatively consider these to be movements of either individual endpoints or sets of at most $B$ endpoints, for a parameter $B$, provided that each set of such endpoints is contained in a given convex region, $R$, and all the endpoints in this region are translated by the same vector. We call such operations the *bulk moves*.

All of our methods are based on our showing interesting connections between the streamed graph drawing problems we study and the file maintenance problem. In addition to utilizing existing algorithms for the file maintenance problem in our graph drawing schemes, we also give a new algorithm for this classic problem in a framework where bulk memory moves are allowed, and we show how this solution can also be applied to streamed graph drawing.

## 2   Building Blocks

*The ordered streaming model.* We start with the description of the model under which we operate. At each time $t \geq 1$, a new edge, $e$, of a graph, $G$, arrives and has to be incorporated immediately into a drawing of $G$, using line segments whose endpoints are placed at grid points with integer coordinates. Since we are focused on planar drawings in this paper, together with the edge, $e$, we also get the information of its relative position among the neighbors of $e$'s endpoints (i.e., for every vertex we know the clockwise order of its neighbors and $e$'s placement in this order). At all times, the current graph, $G$, is connected, and the edges never disappear (infinite persistence).

Incidentally, the streaming model of Binucci *et al.* [4] is slightly different, in that edges arrive without the order information in their model. Under that model, they have given an $\Omega(2^{\frac{n}{8}})$ area lower bound for bi-

nary tree drawing and an $\Omega(n(d-1)^n)$ lower bound for drawing trees with maximum degree $d > 2$. These bounds stand when the algorithm is not allowed to move any vertex. However, they only apply to a very restricted class of algorithms, namely *predefined-location algorithms*, which are non-adaptive algorithms whose behavior does not depend on the order in which the edges arrive or the previously drawn edges. Also, as noted above, Binucci *et al.* do not allow for vertices to move once they are placed. As we show in the following theorem, even with the added information regarding the relative placement of an edge among its neighbors incident on the same vertex, if we don't allow for vertex moves, we must allow for exponential area.

**Theorem 1.** *Under the ordered streaming model without vertex moves, any tree drawing algorithm requires $\Omega(2^{n/2})$ area in the worst case.*

*Proof.* We start with a single node $r$ (*root*) placed in an empty grid and imagine it surrounded by an $2 \times 2$ *bounding square* (see Fig. 1). After adding at most 5 edges from the root (ordering doesn't matter), there has to be one side $m$ of the *bounding square* that is pierced by two edges, $e$ and $f$. Assume w.l.o.g. that $m$ is the upper side of the *bounding square*. $e$ and $f$ form an (infinite) wedge $W$, and their points of intersection with $m$ specify $s$, a sub-interval of $m$. Abusing the notation so that $s$ denotes the length of $s$, we have $s \leq 2$.

We keep adding new edges from the root *inside* (shrinking) $W$. Adding new edge inside $W$ (enforced by specifying edge ordering) divides $W$ into two wedges, $W_1$ and $W_2$. For the next iteration we select the one with the smaller interval $s$. Thus, after adding $n+1$ edges from the root, $s \leq 1/2^n$ in the current wedge.

Now let us limit $W$ with a lowest possible horizontal line $L$ such that there are two grid points inside $W$ that lie on $L$. There are two triangles cut out from $W$ – one (red) by the *bounding square*, the other (blue) by $L$. They are similar, so $A/S = a/s$. Then

$$A = \frac{aS}{s} \geq 2^n aS \geq 2^n,$$

as $a \geq 1$ and $S \geq 1$.

Now look at the large (blue) triangle. By definition, it can contain at most $A$ grid points (at most one for each $y$-coordinate; distance of $r$ from $L$ is $\leq A$). Placing of $\log A \approx n$ additional edges allows us to obtain a sub-wedge with no grid points inside the blue triangle (by always picking the sub-wedge that contains less grid points). Now every new edge has

to be placed at distance at least $A$ from the root as there are no more available grid points left inside the triangle. Clearly, the area needed for the drawing is now also at least $A \geq 2^n$. Since our tree has $2n$ edges, we get an $\Omega(2^{n/2})$ area lower bound for drawing a tree of $n$ edges. $\qquad\square$
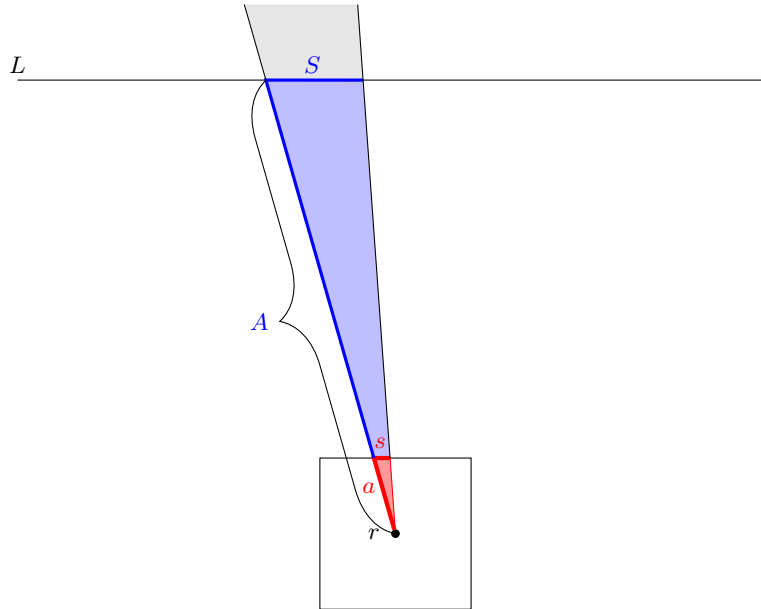


**Fig. 1.** Illustration of the proof of Theorem 1

***File maintenance with bulk moves.*** Here we consider the file maintenance problem and the online list labeling problem variants where we allow for bulk moves[2] of an interval of $B$ labels, for some parameter $B$. We have already mentioned the known results for the file maintenance problem, where the only type of relabelings that are allowed are for individual elements, in which $O(\log^2 n)$ worst-case relabelings suffice for each update when $N$ is $O(n)$ [2,11] and $O(\log n)$ suffice in the worst-case when $N$ is a polynomial in $n$ [7].

Bulk moves allow us to improve on these bounds. We have achieved several tradeoffs between the operation count and the size of $B$. Of course,

---

[2] Note that bulk moves are also motivated for the original file maintenance problem if we define the complexity of a solution in terms of the number of commands that are sent to a DMA controller for bulk memory-to-memory moves.

if $B$ is $n$, then achieving constant number of operations is easy, since we can maintain the $n$ elements to have the indices 1 to $n$, and with each insertion, at some rank $i$, we simply move the elements currently from $i$ to $n$ up by one, as a single bulk move. Theorem 2 summarizes the rest of our results.

**Theorem 2.** *We can achieve the following bounds:*

1. *$O(1)$ worst-case relabeling bulk moves suffice for the file maintenance problem if $B = n^{1/2}$.*
2. *$O(1)$ worst-case relabeling bulk moves suffice for the online list maintenance problem if $B = \log n$.*
3. *$O(\log n)$ worst-case relabeling bulk moves suffice for the file maintenance problem if $B = \log n$.*

*Proof.*

1. This is accomplished in an amortized way by partitioning the array into $n^{1/2}$ chunks of size at most $2n^{1/2}$. Whenever a chunk, $i$, grows to have size $n^{1/2}$, we move all the chunks to the right of $i$ by one chunk (using $O(n^{1/2})$ bulk moves). Then we split the chunk $i$ in two, keeping half the items in chunk $i$ and moving half to chunk $i + 1$. These moves are charged to the previous $n^{1/2}/2$ insertions in chunk $i$. Turning this bound into a worst-case bound is then done using standard de-amortization techniques.
2. This is accomplished by slightly modifying a two-level structure of Kopelowitz [7]. Kopelowitz used the top level of this structure to maintain order of sublists of size $O(\log n)$ each. Order within each sublist was maintained using standard file maintenance problem solutions. Our modification is that each sublist is now represented as a small subarray of size $O(\log n)$ and operations on the top level of Kopelowitz's structure are simulated using bulk moves on a big array containing all concatenated subarrays.
3. This is accomplished by using the method of Bender *et al.* [2] and noting that each insertion in their scheme uses a process where each substep involves moving a contiguous subarray of size $O(\log n)$ using $O(\log n)$ individual moves. Each such move can alternatively be done using $O(1)$ bulk moves of subarrays of size $\log n$.

## 3  Streamed Graph Drawing of Trees

In this section, we present several algorithms for upward grid drawings of trees in the ordered streaming model. The algorithms are designed to

accommodate different types of vertex moves allowed. For example, by a
*bulk move*, we mean a move that translates all segment endpoints that
belong to a given convex region, $R$, by the same vector. This corresponds
to the observation [12] that moving a small number of elements in the
same direction is easy to follow and does not interfere with the ability to
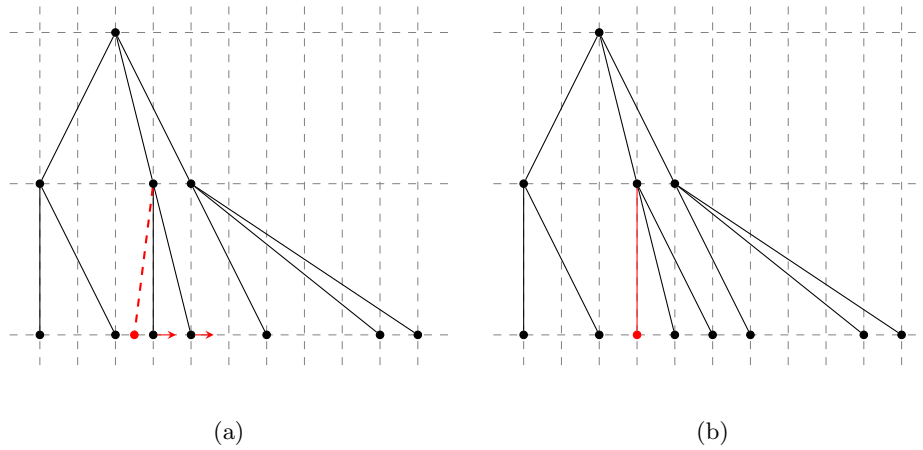understand the structure of the drawing (as long as there are no intersec-
tions).



(a)                                        (b)

**Fig. 2.** Illustrating an insertion for our tree-drawing scheme: (a) determining relative
position for the new (dashed, red) edge; (b) tree after edge insert and related vertex
moves.

Let $G$ be a tree that is revealed one edge at a time, keeping the graph
connected. Algorithm 1 selects one endpoint of the first edge, $r$, puts it
at position $(0,0)$, and produces an upward straight-line grid drawing of
$G$, level-by-level, with each edge from parent to child pointed downwards.
(If a new edge is ever revealed for the current root, we simply recalibrate
what we are calling position $(0,0)$ without changing the position of the
vertices already drawn.) For the $k$th level, $L_k$, with $n_k$ nodes, we place
nodes in positions $(0,-k)$ through $(N,-k)$ in the order of their parents
(to avoid intersections), where $N \geq n_k$. When a new edge is added, we
locate the position (row and position in the row) of the new node and
insert the new node after its predecessor (or before its successor), shifting
other nodes on this level as needed to make room for the new node. (See
Fig. 2.) The details are as shown in Algorithm 1.

**Input:** $e = (a, b)$, the edge to be added; $b$ is the new vertex
1: $k \leftarrow b$'s distance from $r$
2: determine $c$, $b$'s predecessor (or successor) in level $L_k$
3: perform $L_k$.insertAfter$(c, b)$ (or $L_k$.insertBefore$(c, b)$), giving $b$ integer label $L(b)$
4: move vertices whose labels have changed in the previous step
5: place $b$ at $(L(b), -k)$ and draw $e$

**Algorithm 1:** Generic insertion algorithm for upward straight-line grid streamed tree drawing.

Drawing the tree in this fashion ensures there are no intersections (edges connect only vertices in two neighboring levels), even as the vertices are shifted (relative order of vertices stays the same). In addition, there are at most $O(n)$ levels, and at most $O(n)$ nodes per level.

**Theorem 3.** *Depending on the implementation for the insertBefore and insertAfter methods, Algorithm 1 maintains a straight-line upward grid drawing of a tree in the ordered streaming model to have the following possible performance bounds:*

1. *$O(n^2)$ area and $O(1)$ vertex moves per insertion if bulk moves of size $n^{1/2}$ are allowed.*
2. *$O(n^2)$ area and $O(\log n)$ vertex moves per insertion if bulk moves of size $\log n$ are allowed.*
3. *$O(n^2)$ area and $O(\log^2 n)$ vertex moves per insertion if bulk moves are not allowed.*
4. *polynomial area and $O(1)$ vertex moves per insertion if bulk moves of size $\log n$ are allowed.*
5. *polynomial area and $O(\log n)$ vertex moves per insertion if bulk moves are not allowed.*

*Proof.* The claimed bounds follow immediately from Theorem 2. □

Note that $\Omega(n^2)$ area is necessary in the worst case for an upward straight-line grid drawing of a tree if siblings are always placed on the same level.

## 4   Streamed Graph Drawing of Tree-Maps

A tree-map, $M$, is a visualization technique introduced by Shneiderman [9], which draws a rooted tree, $T$, as a collection of nested rectangles. The root, $r$, of $T$ is associated with a rectangle, $R$, and if $r$ has $k$ children, then $R$ is partitioned into $k$ sub-rectangles using line segments parallel

to one of the coordinate axes (say, the $x$-axis), with each such rectangle associated with one of the children of $r$. Then, each child rectangle is recursively partitioned using line segments parallel to the other coordinate axis (say, the $y$-axis). (See Fig. 3.)
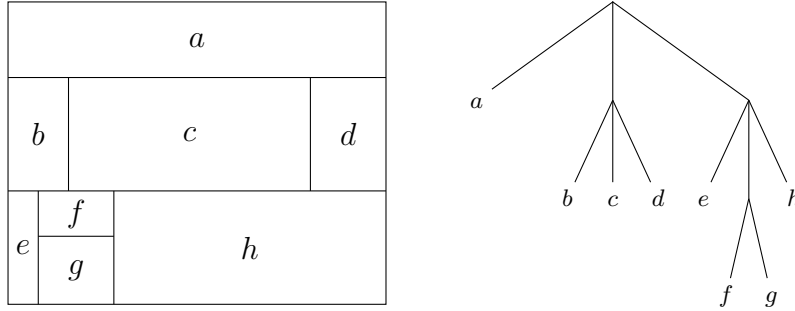


**Fig. 3.** A tree-map and its associated tree.

We assume in this case that a tree, $G$, is released one edge at a time, as in the previous section. We assume inductively that we have a tree-map drawn for $G$, with a global set, $X$, of all $x$-coordinates maintained for the rectangle boundaries and a global set, $Y$, of all $y$-coordinates maintained for the rectangle boundaries. When an edge, $e$, of a rectangle has to be moved, the largest segment containing $e$ is moved accordingly. Our insertion method is shown in Algorithm 2 (for brevity, the case when a vertex has no predecessors among its siblings is omitted).

---

**Input:** $e = (a, b)$, the edge to be added; $b$ is a new child vertex
1: Let $R$ be the rectangle for $a$, and let $z$ be the primary axis for $R$ (w.l.o.g., $z = x$)
2: **if** $b$ has no siblings **then**
3:    $R_b \leftarrow R$ (and give it primary axis orthogonal to $z$)
4: **else if** **then**
5: **else**
6:    determine $c$, $b$'s predecessor sibling (w.l.o.g.), and let $R_c$ be $c$'s rectangle
7:    perform $X$.insertAfter($R_c.x_{\max}, b$), giving $b$ integer label $L(b)$
8:    move segment endpoints whose labels have changed in the previous step
9:    $R_b \leftarrow$ the rectangle in $R$ with left boundary $R_c.x_{\max}$ and right boundary $L(b)$
10: **end if**

---

**Algorithm 2:** Generic insertion algorithm for streamed tree-map drawing.

**Theorem 4.** *Depending on the implementation for the insertBefore and insertAfter methods, Algorithm 2 maintains a tree-map drawing of a tree in the ordered streaming model to have the following possible performance bounds:*

1. *$O(n^2)$ area and $O(1)$ x- and y-coordinate moves per insertion if bulk moves of size $n^{1/2}$ are allowed.*
2. *$O(n^2)$ area and $O(\log n)$ x- and y-coordinate moves per insertion if bulk moves of size $\log n$ are allowed.*
3. *$O(n^2)$ area and $O(\log^2 n)$ x- and y-coordinate moves per insertion if bulk moves are not allowed.*
4. *polynomial area and $O(1)$ x- and y-coordinate moves per insertion if bulk moves of size $\log n$ are allowed.*
5. *polynomial area and $O(\log n)$ x- and y-coordinate moves per insertion if bulk moves are not allowed.*

*Proof.* The claimed bounds follow immediately Theorem 2. □

We leave as an exercise how a similar approach could be used for streamed grid straight-line drawings of binary trees, where the $y$-coordinate of a node depends on its depth and its $x$-coordinate depends on its inorder rank.

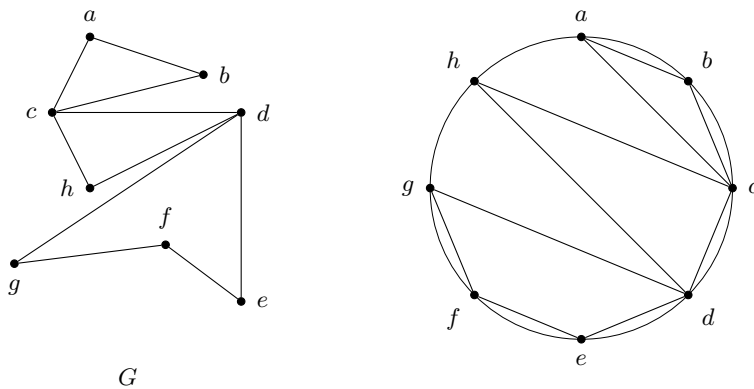## 5   Streamed Graph Drawing of Outerplanar Graphs



**Fig. 4.** An outerplanar graph, $G$, and its circular drawing.

Our algorithm for drawing outerplanar graphs in the streaming model is based on a well-known fact about outerplanar graphs, namely that any outerplanar graph may be drawn with straight-line edges and without intersections in such a way that the vertices are placed on a circle [10]. (See Fig. 4.)

As previously, we assume that each new edge comes with the information about its relative placement among its endpoints' incident edges. In other words, for each vertex, we know the clockwise order of its incident edges. Fig. 5 shows a situation when this information alone is not enough, however.
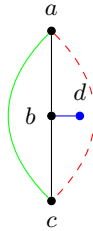


**Fig. 5.** Situation where information about order of edges around vertex is insufficient. Initially, there are vertices $a$, $b$, $c$ and edges $(a, b)$, $(b, c)$. When a new edge $(a, c)$ is added, it can be drawn in two ways (solid green or dashed red) – ordering of edges does not specify which one to choose. When edge $(b, d)$ arrives (with edge order $(a, d, c)$ around $b$), if the dashed red edge location was selected, there is no way to move the vertices without intersections to produce an outerplanar drawing.

Nevertheless, this type of problem can only happen when the new edge connects two vertices of degree 1 as shown below.

**Lemma 1.** *If at least one of the newly connected vertices has degree $> 1$, the information about relative order of incident edges suffice.*

*Proof.* Consider the situation shown in Fig. 6. $(p, q)$ is the new edge ($p$ has degree at least 2). The graph is connected, and the path between $p$ and $q$ is shown. The initial direction of the edge (bold part) is determined by the ordering of edges around $p$. Then the edge can either go around $r$ (shown in dashed red) or not (solid green). Obviously, the dashed red edge location is invalid, as it would surround $r$ with a face, violating the requirement that each vertex belongs to the outer plane. Therefore, there is only one possibility for correctly drawing the edge.    □

It follows that when the new edge connects two vertices of degree 1, additional information (such as relative order of the vertices on the outer face) is necessary.
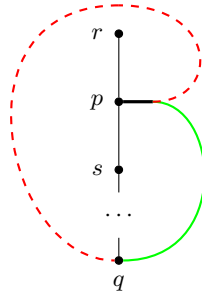
**Fig. 6.** Of the two possibilities of drawing new edge $(p, q)$, only solid green is valid.

We present our streamed drawing algorithm for an outerplanar graph, $G$, in terms of placing vertices of $G$ on a circle. We will later derive an algorithm for drawing $G$ using grid points. We show in Algorithm 3 how to handle adding a new edge to the graph.

---

**Invariant:** vertices are placed on the circle in the same order as they appear on the outer face of the drawing
**Input:** outerplanar drawing of graph $G$ on a circle; $e = (p, q)$ – edge to be added
 1: **if** $q$ is a new vertex **then**
 2:    place $q$ on the circle according to ordering that includes $e$
 3: **else**
 4:    add a *virtual* arc $e'$ connecting $p$ and $q$ according to the specification of $e$ s.t. $e'$ does not intersect any existing edge
 5:    calculate order $O$ of vertices on the outer plane (taking $e'$ into account)
 6:    move vertices into place on the circle according to $O$
 7: **end if**
 8: draw $e$

---

**Algorithm 3:** Adding new edge to outerplanar drawing of graph $G$

As mentioned previously, maintaining the invariant guarantees planarity of the drawing. We now show that vertices can move into place without causing any intersections in the process.

**Lemma 2.** *Moving a vertex $v$ inside the circle along a trajectory that does not intersect any edge non-incident to $v$ does not introduce any intersections and maintains the order of edges around $v$.*

*Proof.* Consider the drawing with edges incident to $v$ removed (marked with dashed lines in Fig. 7). The face to which $v$ belongs (limited by edges and circle boundary) is the area where $v$ can move. Because it is convex, as $v$ moves, its incident edges never intersect the boundaries of the area

(other than at their incident vertices), and the relative order of the edges stays the same.  □



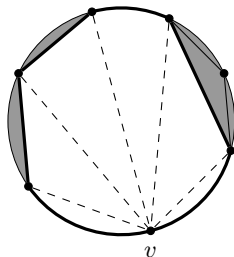**Fig. 7.** Vertex $v$ can move in the (convex) white area without causing intersections or edge order changes.

**Lemma 3.** *A vertex, $v$, that moves into its new position can do so without crossing any edges.*

*Proof.* Consider the face (in the sense of Lemma 2 and Fig. 7) of the drawing that $v$ belongs to. The drawing is still outerplanar after adding the *virtual* arc (which is not necessarily straight-line), and therefore at least part of the circle, $C'$, that forms this face's border still belongs to the outer face of the drawing (see vertex $c$ in Fig. 8). By Lemma 2, $v$ can move to $C'$ without crossing any edges. When there are more vertices whose destination is the same part of the circle, $C''$, (vertices $d$, $e$ and $f$ in Fig. 8), they must form a path with inner vertices all having degree 2. After adding the new edge, their order on the circle (and hence on $C''$) is the reverse of their current order, so their (straight-line) trajectories do not cross. Since they form a path, edges between them will not intersect as they move into place.  □

**Corollary 1.** *Algorithm 3 maintains an outerplanar drawing of a graph $G$ as new edges are added to it.*

Extending Algorithm 3 to placing nodes on a grid is straightforward. Instead of a circle, we operate on a set of grid points in convex position that are approximately circular. We apply one of the algorithms for the file maintenance problem or the online list labeling problem for maintaining order of vertices in this set. When such an algorithm would move
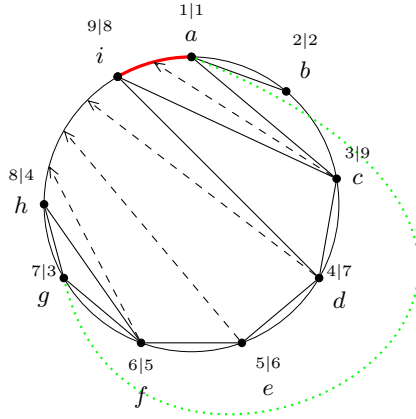
**Fig. 8.** Adding edge $(a, g)$. *Virtual* arc is drawn in green dots. Part of the circle that lies in the outer plane and is reachable from $c$ is shown in bold red. Numbers above vertices denote the order of vertex before and after the edge is added, respectively. Dashed lines are the trajectories of the vertices that need to move to maintain the invariant.

vertex $v$, we first check if there is an unused grid point between new neighbors of $v$ on the circle. If so, we simply move the vertex to that point. Otherwise, we "reserve" the destination for $v$ by inserting a stub vertex in the correct place (between new neighbors of $v$) on the circle. The list labeling algorithm will move vertices around the circle (without changing order on the circle, so it will not cause any intersections) to make room for this stub. Afterwards, we move $v$ into its reserved position.

**Lemma 4.** *Vertex $v$ is moved in line 6 of Algorithm 3 at most $deg(v) - 1$ times.*

*Proof.* A vertex $v$ is moved when the new edge forms a *shortcut* that bypasses $v$ on the outer face. $v$ can appear at most $deg(v)$ times on the outer face, so after $deg(v) - 1$ moves, there will be only one valid position for $v$ on the outer face, so it cannot be bypassed anymore. (See Fig. 9.)

□

**Theorem 5.** *The grid-based version of Algorithm 3 maintains an outer-planar drawing of a graph $G$ and has the following update performances: uses $O(\log n)$ amortized moves per vertex, and*

1. *$O(n^3)$ area and $O(\log^2 n)$ vertex moves per edge insertion.*
2. *polynomial area and $O(\log n)$ vertex moves per edge insertion.*

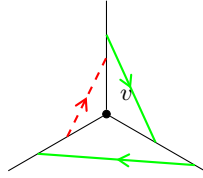*In addition, each of the above complexity bounds applies in an amortized sense per vertex in the drawing.*

**Fig. 9.** Vertex $v$ has degree 3. After two shortcuts (solid green lines) around $v$ have been added, adding a third (dashed red line) would completely surround $v$, violating outerplanarity. Arrows show direction of edges on the outer plane.

*Proof.* By Corollary 1, we know that the algorithm maintains an outerplanar drawing of $G$. For the area bounds, the file maintenance algorithms requires $O(n)$ available integer tags (in this case, points in convex position) to handle $n$ elements. Since $m$ grid points in (strict) convex position require $O(m^3)$ area, the streamed drawing algorithm therefore uses $O(n^3)$ area in such cases. Likewise, it uses polynomial area when using a solution to the online list labeling problem.

With respect to the claim about amortized performance, by Lemma 4, each vertex $v$ is moved by Algorithm 3 at most $deg(v)-1$ times. Each such move requires at most one insertion into the list for the file maintenance or list maintenance algorithm. This means that there are at most $O(n)$ such insertions (sum of degrees of all vertices in an outerplanar graph is $O(n)$). For $O(n)$ insertions, the performance for each of the file maintenance or list maintenance algorithms therefore become an amortized number of moves per vertex made by our algorithm. □

Note that in this application we cannot immediately apply our results for bulk moves, unless we restrict our attention to possible vertex points that are uniformly distributed on a circle and moves that involve rotations of intervals of points around this circle.

## 6   Conclusion

In this paper, we provide a revised approach to streamed graph drawing based on utilizing solutions to the file maintenance problem, either on a level-by-level basis (for level drawings of trees), a cross-product basis (for tree-maps), or a circular/convex-position basis (for outerplanar graphs). For future work, it would be interesting to find other applications of this problem in streamed or dynamic graph drawing applications.

## Acknowledgements

## References

1. D. Archambault and H. C. Purchase. Mental map preservation helps user orientation in dynamic graphs. In W. Didimo and M. Patrignani, editors, *Graph Drawing*, volume 7704 of *LNCS*, pages 475–486. Springer, 2013.
2. M. A. Bender, R. Cole, E. D. Demaine, M. Farach-Colton, and J. Zito. Two simplified algorithms for maintaining order in a list. In R. H. Möhring and R. Raman, editors, *ESA*, volume 2461 of *LNCS*, pages 152–164. Springer, 2002.
3. C. Binucci, U. Brandes, G. D. Battista, W. Didimo, M. Gaertler, P. Palladino, M. Patrignani, A. Symvonis, and K. Zweig. Drawing trees in a streaming model. *Information Processing Letters*, 112(11):418–422, 2012.
4. C. Binucci, U. Brandes, G. Di Battista, W. Didimo, M. Gaertler, P. Palladino, M. Patrignani, A. Symvonis, and K. A. Zweig. Drawing trees in a streaming model. In D. Eppstein and E. R. Gansner, editors, *Graph Drawing*, volume 5849 of *LNCS*, pages 292–303. Springer, 2009.
5. P. Dietz and D. Sleator. Two algorithms for maintaining order in a list. In *19th ACM Symp. on Theory of Computing (STOC)*, pages 365–372, 1987.
6. P. Eades, W. Lai, K. Misue, and K. Sugiyama. Preserving the mental map of a diagram. *Proceedings of Compugraphics*, pages 24–33, 1991.
7. T. Kopelowitz. On-line indexing for general alphabets via predecessor queries on subsets of an ordered list. In *FOCS*, pages 283–292. IEEE Computer Society, 2012.
8. K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210, 1995.
9. B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph.*, 11(1):92–99, Jan. 1992.
10. R. Tamassia. *Handbook of Graph Drawing and Visualization (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, 2007.
11. D. E. Willard. Good worst-case algorithms for inserting and deleting records in dense sequential files. In *ACM SIGMOD*, pages 251–260, 1986.
12. S. Yantis. Multielement visual tracking: Attention and perceptual organization. *Cognitive psychology*, 24(3):295–340, 1992.